On the Algorithms of Guruswami-Sudan List Decoding over Finite Rings

Guillaume Quintin

Thèse
présentée pour obtenir le grade de
Docteur de l'École polytechnique

Spécialité :
Mathématiques – Informatique

par
Guillaume Quintin

Sur l'algorithme de décodage en liste de Guruswami-Sudan sur les anneaux finis

Soutenue le 22 novembre 2012 devant le jury composé de

Rapporteurs

M. Peter Beelen	Technical University of Denmark
M. Damien Stehlé	École normale supérieure de Lyon

Directeur de thèse

M. Daniel Augot	INRIA Saclay

Codirecteur de thèse

M. Grégoire Lecerf	CNRS & École polytechnique

Examinateurs

M. Jean-Claude Belfiore	Télécom ParisTech
M. Philippe Gaborit	Université de Limoges
M. Antoine Joux	Université de Versailles
M. Jean-Pierre Tillich	INRIA Rocquencourt

Remerciements (Acknowledgments)

Je tiens tout d'abord à remercier mes deux directeurs de thèse sans qui ce travail n'aurait pas été possible. Je remercie Daniel Augot pour la très grande autonomie qu'il m'a laissée durant ces trois dernières années, sa patience et ses conseils concernant les mathématiques et l'informatique. Grégoire Lecerf a, quant à lui, répondu (et réponds toujours !) avec précision et patience à mes questions aussi bien mathématiques qu'informatiques qui sont souvent bêtes.

Je remercie Damien Stehlé et Peter Beelen pour avoir accepté de rapporter cette thèse. Merci pour m'avoir amené dans un bouchon lyonnais. Je remercie Phillipe Gaborit, mon ancien professeur de l'université de Limoges qui m'a beaucoup appris et qui m'a soutenu au début et pendant ma thèse. Je remercie Jean-Claude Belfiore, Antoine Joux et Jean-Pierre Tillich pour avoir accepté de faire partie de mon jury de thèse.

Je remercie l'INRIA et la DGA pour le financement de ma thèse ainsi que l'École polytechnique et le CNRS pour avoir mis à disposition leurs infrastructures. Je remercie tout le laboratoire d'informatique où j'ai été bien accueilli et où j'ai pu rencontrer des gens formidables et toujours très sympathiques.

Je remercie l'équipe CRYPTO du LIX qui m'a accueilli et avec qui j'ai pu apprendre et échanger des mathématiques. Merci à Daniel Augot, François Morain, Ben Smith, Alain Couvreur, Luca de Feo, Jean-François Biasse, Morgan Barbier, Jérôme Milan, Tania Richmond, Cécile Gonçalves, Julia Pieltant, Nicolas Delfosse qui ne s'est toujours pas rendu en chambre 17 et Évelyne Rayssac.

J'ai passé beaucoup de temps, pour ne pas dire "squatté" dans l'équipe de calcul formel du LIX, l'équipe MAX, où j'ai appris des mathématiques et profité de la bibliothèque. Merci donc à Marc Giusti, Joris van der Hoeven, Grégoire Lecerf, Romain Lebreton, Jérémy Berthomieu, François Ollivier et Antoine Colin.

Je tiens aussi à remercier les gens que j'ai vu occasionnellement comme Alexander Zeh, Antonia Wachter-Zeh, Matthieu Legeay, Alexandre Gordien et Sofiane Amari.

Je remercie l'équipe `sysres` du LIX pour leur bonne humeur. James Regis, Matthieu Guionnet, Elie Mabo et Jean-Marc Notin m'ont permis de maîtriser, d'installer et de travailler confortablement sur mon ordinateur "julien" malgré des logiciels datant des années 70 à peine capables de reconnaître une clé USB.

Un grand merci à Jéjé qui m'a appris beaucoup en informatique. Nos maintes discussions m'ont permis en autre de développer ma librairie, de découvrir beaucoup de programmes et de diminuer mon très mauvais goût en "esthétique industrielle".

Je remercie aussi Évelyne Rayssac, Corinne Poulain, Valérie Lecomte, Christelle Liévin, Sylvie Jabinet, Catherine Bensoussan qui m'ont aidé à faire face aux trop nombreux problèmes administratifs avec gentillesse et patience.

Je remercie également Albert Cohen avec qui j'ai partagé les TDs de INF422 et INF583 pendant deux années. Il a toujours répondu avec précision et patience à mes questions sur le `C`, la `libc`, `gcc` et le noyau Linux.

Un énorme merci à mes coburreaux qui ont dû me supporter pendant trois ans. Je pense à Van Du TRAN Thuong, Morgan Barbier et Alain Couvreur qui ont toujours été là pour m'épauler et me conseiller en informatique, géométrie et théorie des codes correcteurs.

Je remercie aussi tous les membres de l'association non officielle TrolLIX (trolls au LIX) pour leur bonne humeur et les bons moments passés au Safran devant coca-cola et muffin au chocolat. Parmi les membres on compte Olivier Schwander (débianneux emacsien), Jérôme Milan (archeux emacsien), François Poulain (débianneux vimiste), Morgan Barbier (archeux emacsien), James Régis (centosien vimiste), Matthieu Guionnet (centosien vimiste) et Jean-Marc Notin (archeux emacsien). Il est bien évident que les vimistes vaincront.

Enfin je remercie vivement mes proches qui ont su faire preuve de beaucoup de patience et de compréhension : Amélie, mes parents Patrick et Marie-Claude Quintin, ma soeur et mon beau frère Amandine et Daniel, ma belle-famille Thierry, Françoise et Guillaume, Sylvie et Dominique, et mes amis Frédéric, Adrien, Claude et Florence, Olivier et Sébastien, Jo et Audrey, Laurie et Greg, Alicia et Willy, Amandine, Clarinnette. Les trois semaines de vacances à Boujassac m'auront fait un très grand bien.

Merci aussi à tous ceux que je n'ai pas cités dans ces deux pages mais qui se reconnaîtront et ne m'en voudront pas.

Contents

List of Algorithms

List of Figures

Organization of the document

This PhD thesis is structured in four parts. Each part contains a small self-contained introduction. It recalls some definitions that will help read the chapters and contains a small paragraph which briefly describes the contributions contained within the part. The parts are independent from each other.

The chapters are in fact submitted or accepted papers. They have not been modified from their original version. They are therefore self-contained and independent from each other and can be read in any order.

Part I and Part II contain two concurrent approaches for decoding Reed-Solomon codes over finite rings while Part IV describes the implementation of certain algorithms presented in the first two parts. Part III contains two chapters presenting results obtained during my PhD thesis which are not directly related to the decoding of Reed-Solomon codes. The chapters contained in Part III are also self-contained and can be read in any order and even before the other three parts of this PhD thesis.

Introduction

Context

Error correcting codes and their unambiguous decoding

The theory of error correcting codes initiated by the work ofShannon [Sha48] and Hamming [Ham50] deals with the correction of errors during data transmissions over noisy channels to obtain reliable communications. In this PhD thesis we are interested in algebraic error correcting codes which form a sub-field of the general theory of correcting codes.

Fix a finite set E also called an *alphabet*. We denote by $|E|$ its cardinality. An element of E is also called a *symbol* of E. Suppose that one wants to send over a noisy channel (Internet, radio waves) a message constituted of m symbols where m is a positive integer. First he will cut the message into m/n *words* of n symbols (an element of E^n). The recipient of the message will obtain distorted words. They are also elements of E^n but they are not the original words as some symbols could have changed. One way among other to measure the difference between the received word and the original word is by mean of a distance on E^n. Of course one can also send words whose each symbol belongs to a different alphabet. Hence the words belong to $E_1 \times \cdots \times E_n$ where E_i is an alphabet for $i = 1, \ldots, n$ and, *a priori*, $E_i \neq E_j$ for $i \neq j$. In this PhD thesis we will only consider the *Hamming distance*.

Definition 1 (Hamming weight and distance)**.** Let $E_1, \ldots, E_n$ be (possibly infinite) sets and

$$x = (x_1, \ldots, x_n) \in \prod_{i=1}^{n} E_i, y = (y_1, \ldots, y_n) \in \prod_{i=1}^{n} E_i.$$

The *Hamming distance* between x and y, denoted by $d(x, y)$ in this manuscript, is

$$\left|\{i \in \{1, \ldots, n\} : x_i \neq y_i\}\right|.$$

Suppose that E_i is a monoid with neutral element 0_{E_i} for $i = 1, \ldots, n$. The *Hamming weight* of x, denoted by $w(x)$, is

$$\left|\{i \in \{1, \ldots, n\} : x_i \neq 0_{E_i}\}\right|$$

and we have, if E_i is a group, $d(x, y) = w(x - y)$.

The Hamming distance take only into account the number of symbols which differ from the original ones. Of course, other distances can be considered. The *Lee distance*, for example, like the Hamming distance, measures the number of symbols which differ from the original ones but also how much they differ. The *weighted Hamming distance* is used for example to give more importance to certain symbols than others.

Definition 2. Let $E_1, \ldots, E_n$ be (possibly infinite) sets, $\alpha = (\alpha_1, \ldots, \alpha_n) \in \mathbb{R}^n$ be such that $\alpha_i > 0$ for all $i = 1, \ldots, n$ and

$$x = (x_1, \ldots, x_n) \in \prod_{i=1}^{n} E_i, y = (y_1, \ldots, y_n) \in \prod_{i=1}^{n} E_i.$$

The *weighted Hamming distance* between x and y is defined to be

$$d_\alpha(x, y) := \sum_{i=1}^{n} \alpha_i \delta(x_i, y_i)$$

where $\delta(x, y) = 0$ if $x = y$ and 1 otherwise. Suppose that E_i is a monoid with neutral element 0_{E_i} for $i = 1, \ldots, n$. The *Hamming weight* of x is defined to be

$$w_\alpha(x) := \sum_{i=1}^{n} \alpha_i \delta(x_i, 0_{E_i})$$

and we have $d_\alpha(x, y) = w_\alpha(x - y)$. Note that when $\alpha = (1, \ldots, 1)$ we get the Hamming distance, $d = d_\alpha$.

An *error correcting code* is a subset of words of E^n or $E_1 \times \cdots \times E_n$. In fact we can represent it as a monomorphism. Informally speaking it is the same as taking k symbols, with $k < n$, adding $n - k$ redundancy symbols to obtain a word with n symbols then sending it over the channel.

Definition 3. We take the notation of Definition 1. Let $k < n$ be two positive integers. An injection of sets

$$\varphi : E_{i_1} \times \cdots \times E_{i_k} \longrightarrow E_1 \times \cdots \times E_n$$

where $\{i_1, \ldots, i_k\} \subset \{1, \ldots, n\}$ is called an *error correcting code* or simply a *code of blocklength* n. When the context is clear we will simply call error correcting code the subset $\mathcal{C} = \varphi(E_{i_1} \times \cdots \times E_{i_k})$ of $E_1 \times \cdots \times E_n$. We will also call the blocklength of $\mathcal{C}$ its *length*. The *rate* of $\mathcal{C}$ is defined to be k/n. The *minimum (Hamming) distance* of $\mathcal{C}$ is defined as

$$\min \{d(x, y) : x, y \in \mathcal{C} \text{ and } x \neq y\}.$$

The elements of $\mathcal{C}$ are called *codewords*. Let A be any ring with identity. If E_i for all $i = 1, \ldots, n$ is an A-module and φ is a morphism of A-module, then $\mathcal{C}$ is called a *linear code over* A.

Notation 4. Taking the notation of Definition 3 we say that $\mathcal{C}$ has *parameters* (n, k, d). If $\mathcal{C}$ is a linear code we denote its parameters by $[n, k, d]_A$.

Given a code $\mathcal{C}$ of length n and minimal distance d, it is obvious that two balls of radius

$$t = \left\lfloor \frac{d-1}{2} \right\rfloor$$

centered in two different codewords are disjoint. This interesting property allows the recipient of a word to correct it into a codeword as soon as the received word is within a ball of radius t centered in a codeword. This simple fact is at the origin of many decoding algorithms.

Definition 5. Let $\mathcal{C}$ be a code such that $\mathcal{C} \subset E_1 \times \cdots \times E_n$. An map of sets

$$\psi : E_1 \times \cdots \times E_n \longrightarrow \mathcal{C} \cup \{?\}$$

where $? \notin \mathcal{C}$ is called a *unique decoding function of* $\mathcal{C}$. The "?" element means that ψ is unable to correct a word into a codeword. Note that a unique decoding function that can correct up to t errors is the following:

$$\begin{array}{rcl} u_{\mathcal{C}} : E_1 \times \cdots \times E_n & \longrightarrow & \mathcal{C} \cup \{?\} \\ x = (x_1, \ldots, x_n) & \longmapsto & \left\{ \begin{array}{ll} c & \text{if } x \text{ is within a ball of radius } t \text{ centered in } c, \\ ? & \text{otherwise.} \end{array} \right. \end{array}$$

The *decoding radius* of ψ is the greatest positive integer τ such that $\psi(x) \neq ?$ whenever x is within a ball of radius τ centered in a codeword of $\mathcal{C}$. The decoding radius of $u_{\mathcal{C}}$ is at least t. We will also call *decoding radius of* ψ any integer $\tau' \leq \tau$.

Actually given a code, it is difficult to design an algorithm which is able to correct up to t errors. It is the case, for example, of the Bose, Ray-Chaudhuri and Hocquenghem (BCH) codes. This family of codes is constructed in such a way that one only knows a lower bound on their minimal distances and the unique decoding function for such a code is able, *a priori*, to correct strictly less than t errors [MS86a, Chapter 9, Research Problem 9.3, page 277].

An ideal code $\mathcal{C}$, for a practical use, would be a code with a high rate (near 1), a big minimal distance d as the decoding radius of $u_{\mathcal{C}}$ grows with d, and an efficient unique decoding function whose decoding radius is t. But the first two wishes are incompatible.

Proposition 6 (Singleton bound [MS86a, page 544])**.** *Suppose that E is a finite set with $q := |E|$ and let $\mathcal{C} \subseteq E^n$ be a code of parameters (n, k, d). Then $|\mathcal{C}| \leq q^{n-k+1}$.*

Fortunately a family of linear codes was discovered in 1960 by Irvin Stoy Reed and Gustave Solomon in their original paper [RS60]. Any code from this family of parameters $[n, k, d]$ is such that $d = n - k + 1$. They are obviously called *Reed-Solomon* codes.

Definition 7 (Maximum distance separable codes)**.** Any code of minimal distance $n - k + 1$ is called a *maximum distance separable* (MDS) code.

We give the definition of Reed-Solomon codes over finite fields as it is the most common situation in practice [WB99]. They can be defined over any commutative field.

Definition 8 (Classical Reed-Solomon codes)**.** Let $k < n$ be two positive integers and $x = (x_1, \ldots, x_n) \in \mathbb{F}_q^n$ be such that for all $i \neq j$, $x_i \neq x_j$. The vector subspace generated by the

$$(f(x_1), \ldots, f(x_n)) \in \mathbb{F}_q^n, f \in \mathbb{F}_q[X] \text{ and } \deg f < k$$

is called a *Reed-Solomon code of parameters* $[n, k]_{\mathbb{F}_q}$ and denoted by $\mathrm{RS}_{\mathbb{F}_q}(x, n)$ or simply $\mathrm{RS}(n, k)$ when there is no confusion. The vector x is called the *support* of $\mathrm{RS}_{\mathbb{F}_q}(x, k)$.

There exist several efficient decoding algorithms for Reed-Solomon codes than can correct up to

$$t = \left\lfloor \frac{d-1}{2} \right\rfloor = \left\lfloor \frac{n-k}{2} \right\rfloor$$

errors, for example [Pet60, Jus76, BW86, Gao02]. There is also the Berlekamp-Massey algorithm [Ber68, Section 7.3], [Mas69] and the Euclid-based algorithms [SKHN75], [Bla83, Chapter 7], [TERH88]. A bigger and important family of linear codes, which include all Reed-Solomon codes, shares the same properties. They are called *generalized Reed-Solomon codes*.

Definition 9 (Classical generalized Reed-Solomon codes)**.** Let $k < n$ be two positive integers, $v = (v_1, \ldots, v_n)$ be such that $v_i \neq 0$ and $x = (x_1, \ldots, x_n) \in \mathbb{F}_q^n$ be such that for all $i \neq j$, $x_i \neq x_j$. The vector subspace generated by the

$$(v_1 f(x_1), \ldots, v_n f(x_n)) \in \mathbb{F}_q^n, f \in \mathbb{F}_q[X] \text{ and } \deg f < k$$

is called a *generalized Reed-Solomon code of parameters* $[n, k]_{\mathbb{F}_q}$ and denoted by $\mathrm{GRS}_{\mathbb{F}_q}(x, n)$ or simply $\mathrm{GRS}(n, k)$ when there is no confusion. The vector x is called the *support* and the vector v is called the *weight* of $\mathrm{GRS}_{\mathbb{F}_q}(x, k)$.

This family of codes allows to construct in a way I will describe later the important family of BCH codes which are themselves a sub family of cyclic codes [MS86a, Chapter 7].

Definition 10 (Cyclic codes)**.** Let $\mathcal{C} \subset E^n$. We say that $\mathcal{C}$ is *cyclic* if

$$(c_1, \ldots, c_{n-1}, c_n) \in \mathcal{C} \Longrightarrow (c_n, c_1, \ldots, c_{n-1}) \in \mathcal{C}.$$

Theorem 11. *Taking the notations of Definition 10, let $\mathcal{C}$ be a cyclic linear code over $E = \mathbb{F}_q$ (the finite fields with q elements). Then the $\mathbb{F}_q$-linear map*

$$\begin{array}{rcl} \mathbb{F}_q^n & \longrightarrow & \mathbb{F}_q[X]/(X^n - 1) \\ c = (c_1, \ldots, c_n) & \longmapsto & c_1 + c_2 X + \cdots + c_n X^{n-1} \end{array}$$

is an isomorphism. Moreover this isomorphism induces a one-to-one correspondence between cyclic codes over $\mathbb{F}_q$ of length n and ideals of $\mathbb{F}_q[X]/(X^n - 1)$.

As $\mathbb{F}_q[X]/(X^n-1)$ is a principal ideal ring, every ideal is generated by one element called the *generator* of the corresponding cyclic codes.

Definition 12 (BCH codes)**.** A BCH code over $\mathbb{F}_q$ is a cyclic code over $\mathbb{F}_q$ such that the set of roots of its generator polynomial contains

$$\{\alpha^b, \alpha^{b+1}, \ldots, \alpha^{b+\delta-2}\}$$

for $\alpha \in \mathbb{F}_q^a$ and b, δ positive integers. The positive integer δ is called the *designed minimum distance of* $\mathcal{C}$.

If $\mathcal{C}$ denotes a BCH code over $\mathbb{F}_q$ of length n and designed minimum distance δ, it can be shown that there exists a finite extension $\mathbb{F}_{q^s}$ of $\mathbb{F}_q$ and a generalized Reed-Solomon code $\mathcal{G} = \mathrm{GRS}_{\mathbb{F}_{q^s}}(n, n-\delta+1)$ such that $\mathcal{C} = \mathcal{G} \cap \mathbb{F}_q^n$. Therefore any unique decoding function for $\mathcal{G}$ will induce a unique decoding function for $\mathcal{C}$. Note also that when $n = q^s - 1$, $\mathcal{C}$ is in fact a Reed-Solomon code. A consequence is that any Reed-Solomon code over $\mathbb{F}_q$ of length $q-1$ is also a cyclic code. For more details see for example [MS86a, Chapters 3, 7, 9, 10 and 12].

We finish this subsection with definition of *erasures*. Let E_i be an alphabet for all $i = 1, \ldots, n$ and let "?" be such that $? \notin E_i$ for all $i = 1, \ldots, n$. We suppose that the channel used by the sender does not distort words but instead "loses" some symbols. This is the case for example over Internet when a TCP connection is established. The packets of a TCP stream are numbered, they are not corrupted but some packets can be lost. The recipient of the message will then receive a word $(x_1, x_2, ?, x_3, ?, ?, x_6) \in E^6$ provided that $(x_1, x_2, x_3, x_4, x_5, x_6)$ was sent over the channel.

Definition 13 (Erasures)**.** Let E_i be an alphabet for all $i = 1, \ldots, n$ and let "?" be such that $? \notin E_i$ for all $i = 1, \ldots, n$. Let $\mathcal{C} \subset E_1 \times \cdots \times E_n$ be a code. We say that a channel is an *erasures channel* if for any received word

$$(x_1, \ldots, x_n) \in (E_1 \cup \{?\}) \times \cdots \times (E_n \cup \{?\})$$

there exists a codeword $(c_1, \ldots, c_n)$ such that for all $i = 1, \ldots, n$, either $x_i = c_i$ or $x_i =?$. Informally speaking we call *erasure* the symbol "?". We do not give any formal definition of a channel in this PhD thesis. We refer the reader to [CT06, Chapter 7]. In this situation we also call *unique decoding function* any function

$$\psi : (E_1 \cup \{?\}) \times \cdots \times (E_n \cup \{?\}) \longrightarrow \mathcal{C} \cup \{?\}.$$

Erasures are easier to handle than classical errors. The recipient knows where they are, and he knows that received (non erased) symbols have not been distorted by the channel. Concerning Reed-Solomon codes over a field the decoding algorithm reduces to univariate polynomial interpolation. If the Reed-Solomon code has parameters $[n, k, d]$ then it can correct up to $n-k$ erasures.

List decoding of error correcting codes

We now focus on the main topic of this PhD thesis, the correction of errors. We can extend the definition of a decoding function.

Definition 14. Let $\mathcal{C}$ be a code such that $\mathcal{C} \subset E_1 \times \cdots \times E_n$ and let "?" be such that $? \notin \mathcal{C}$, $? \notin E_i$ for all $i = 1, \ldots, n$. A map of sets

$$\psi : (E_1 \cup \{?\}) \times \cdots \times (E_n \cup \{?\}) \longrightarrow \mathcal{P}(\mathcal{C}) \cup \{?\}$$

where $\mathcal{P}(\mathcal{C})$ designates the set of all the subsets of $\mathcal{C}$, is called a *list decoding function for* $\mathcal{C}$. The *decoding radius* of ψ is the greatest positive integer τ such that $c \in \psi(x)$ for all codewords c within the ball of radius τ centered in x. We will also call *decoding radius of* ψ any integer $\tau' \leq \tau$.

Note the difference with a unique decoding function which focuses on the Hamming balls centered in the codewords while a list decoding function looks at the ball centered in the received word (which is, *a priori*, not a codeword). The idea behind list decoding is to correct more errors than unique decoding algorithms, typically, to correct more than half the minimum distance errors. Let $\mathcal{C}$ be a code of parameters $[n, k, d]$. Two balls centered in two different codewords of radius greater

$$t = \left\lfloor \frac{d-1}{2} \right\rfloor$$

can intersect and then a list decoding function can give more than one codeword. Fortunately, this is a very rare event [MS86b, RU10], making list decoding usable in practice. Even in the situation where the list decoding function returns at least two errors, it is not worse than returning a failure ("?"), the codewords received before and after the ambiguous word can help pick up the sent codeword. It is the case, for example, when actual words of a spoken language are transmitted.

List decoding has been considered in the 1950s [Eli57, Woz58]. Existence result indicating that list decoding can correct many more errors than unique decoding are known since the 1980s [ZP81, Eli91]. Algorithmic results for algebraic codes motivated by complexity theory then appeared [GL89, Dum89, ALRS92, Sid94, ALRS98]. Then M. Sudan proposed a polynomial time algorithm for Reed-Solomon codes than can correct significantly more errors that half the minimum distance [Sud97a], further improved by V. Guruswami and M. Sudan [GS98]. For general considerations and applications to computer science about list decoding I refer the reader to [Gur10] and the many pointers in it. We review now the so called Guruswami-Sudan algorithm principle. We let $\mathcal{C}$ be a Reed-Solomon code over $\mathbb{F}_q$ of parameters $[n, k, d = n - k + 1]_{\mathbb{F}_q}$.

The first step is often called the *interpolation step*. It consists in finding an algebraic curve passing through certain points of $\mathbb{A}^2(\mathbb{F}_q)$ with given multiplicities. Several algorithms exist [Köt96, OS99, NH00, KV03, Ale05, AZ08] but they have not been implemented or, at least, are not available through a library or a computer algebra system. The second step, also called the *root finding* step is a standard, well known, computer

Algorithm 1 Overview of the Guruswami-Sudan algorithm.

Input: A received word $y \in \mathbb{F}_q^n$ and a decoding radius τ.

Output: All codewords $c \in \mathcal{C}$ such that $d(c, y) \leq \tau$.

1: Find a bivariate polynomial $Q(X, Y) \in \mathbb{F}_q[X, Y]$ satisfying certain properties.

2: Find all the roots of $Q(X, Y)$ seen as a univariate polynomial of $(\mathbb{F}_q[X])[Y]$.

algebra problem. It has been implemented for example in the `Mathemagix` computer algebra system [H+02], in `Magma` [BCP97] and in `Maple` [Map12]. It has been studied in the context of list decoding in [RR98, GS00].

To the knowledge of the author there is no available software providing any list decoding algorithms except the `Percy++` library [Gol07b]. However it is not the main goal of the library and the implementation of the Guruswami-Sudan algorithm is not optimized and is not directly accessible. Computer algebra systems, like `Magma`, which provides coding theory related functions and algorithms, only propose unique decoding algorithms. In fact, `Magma` does not allow the construction of Reed-Solomon codes other than cyclic Reed-Solomon codes.

The decoding radius of the Guruswami-Sudan algorithm is equal to

$$\left\lceil n - \sqrt{n(k-1)} \right\rceil - 1.$$

This value is often called *Johnson bound* for Reed-Solomon codes.

Definition 15. Let $\mathcal{C} \subset E_1 \times \cdots \times E_n$ be a code such that $q_i := |E_i| < +\infty$ for all $i = 1, \ldots, n$ and J a positive integer. We say that J is a *Johnson bound* for $\mathcal{C}$ if all balls of radius J contain at most a number of $(n \max(q_1, \ldots, q_n))^\gamma$ codewords, for a positive integer γ.

It can be shown [Gur04, Theorem 7.10, page 163] that

$$J = n - \sqrt{n(n-d)} \tag{1}$$

is a Johnson bound for all error correcting codes of parameters (n, k, d). In this manuscript we will only consider the Johnson bound given by equation (1) and thus call it *the Johnson bound*. Informally speaking, it says that there are not too many codewords in any ball, in fact, at most a polynomial number of codewords. Therefore we wish list decoding algorithms whose decoding radius is at most J to run in polynomial time.

Other Johnson bounds exist. The above Johnson bound does not take into account the sizes of the alphabets. In [Gur04, Theorem 3.2, page 35] other Johnson bounds are presented. They depend on the sizes of the E_i and/or on the number of wanted codewords in a ball of radius J. Johnson bounds for other metrics are known, for example with the Lee metric [Rot06, page 330 and 331] and [Ber84, Chapter 13]. For more details about Johnson bounds see [Gur04, Chapter 3 and 7] and the pointers in it.

The decoding radius of the Guruswami-Sudan algorithm can thus reach the Johnson bound as, for Reed-Solomon codes, we have $n-d = k-1$. Moreover it runs in polynomial time in $n, \log q, k$.

Algebraic prerequisites

We review in this subsection some basics of algebra and explain what kind of rings are studied in each chapter.

Much more on algebra can be found, for example, in [Lan02, AM94, Mat80, MRS01]. We assume that the reader is familiar with the notion of group, commutative and non commutative rings.

Definition 16. Let A be a ring. A *left ideal* (resp. *right ideal* of A is a subgroup $I \subseteq A$ such that $AI \subseteq I$ (resp. $IA \subseteq I$). If $AI = IA$, we say that I is a *two-sided* ideal.

Definition 17. Let A be a ring whose only ideal is (0). We say that A is a field.

Let A be a ring and E be a subset of A. We denote by (E) the ideal of A consisting of all the *finite* sums

$$\sum a_i x_i$$

where $a_i \in A$ and $x_i \in E$. We call the elements of E *generators* of $I = (E)$. Note that if A is finite, then any ideal can be generated by a finite number of elements.

Definition 18. Let A be a commutative ring and I be an ideal of A. We say that I is a *maximal ideal* if A/I is a field. We say that I is a *prime ideal* if A/I is a domain. A commutative ring containing only one maximal ideal is called a *local ring*.

Example 19. The ring of integers $\mathbb{Z}$ is a domain but is not local. The ring of rationals $\mathbb{Q}$ is a domain and is local. The ring of polynomials $\mathbb{Q}[X]$ is also a domain and not local. The ring of power series $\mathbb{Q}[[X]]$ is a domain and local.

In this PhD thesis we will mainly study local rings whose maximal ideal can be generated by one element.

Definition 20. Let A be a commutative local domain which is not field. If all the ideals of A can be generated by one element we say that A is a *discrete valuation ring* (DVR). Any generator of $\mathfrak{m}$ is called a *uniformizing parameter of* A. For any element $x \in A$ and $x \neq 0$, we denote by $v(x)$ the greatest integer i such that $x \in \mathfrak{m}^i \setminus \mathfrak{m}^{i+1}$ and we set $v(0) = +\infty$. We call the integer $v(x)$ the *valuation of* x. It is easy to see that $\mathfrak{m} = \{x \in A : v(x) \geq 1\}$.

We give two important examples of discrete valuation rings for this PhD thesis. They are constructed as extensions of discrete valuations rings.

Theorem 21 (See [Lan02, Paragraph 4, Chapter 12])**.** *Let A be a discrete valuation ring with uniformizing parameter π and a polynomial $f(X) \in A[X]$ of degree d such that $f \bmod \pi \in (A/\mathfrak{m})[X]$ is irreducible. Then the ring $L = \frac{A[X]}{(f(X))}$ is also a discrete valuation ring such that $A \subseteq L$ with uniformizing parameter π.*

Example 22. Let $\mathbb{F}_q$ be the finite field with q elements and consider $A = \mathbb{F}_q[[t]]$. Then if $f(X) \in A[X]$ has degree d and is irreducible modulo t, we have the following ring isomorphisms:

$$\frac{\mathbb{F}_q[[t]][X]}{(f(X))} \approx \mathbb{F}_{q^d}[[t]] \text{ and } \frac{\mathbb{F}_q[[t]]/(t^r)[X]}{(f(X) \mod t^r)} \approx \mathbb{F}_{q^d}[[t]]/(t^r).$$

Also, if p designates a prime, $\mathbb{Z}_p$ the ring of p-adic integers and $f(X) \in \mathbb{Z}_p[X]$ of degree d such that $f \mod p$ is irreducible. Then, if we denote $\frac{\mathbb{Z}_p[X]}{(f(X))}$ by $\mathbb{Z}_{p^d}$ we have the ring isomorphism

$$\frac{\mathbb{Z}_{p^d}}{(p^r)} \approx \frac{\mathbb{Z}/p^r\mathbb{Z}[X]}{(f(X) \mod p^r)}.$$

Informally speaking, an element of the ring $\mathbb{Z}_{p^d}$ is a power series in p whose coefficients are elements of the extension field $\mathbb{F}_{p^d} = \mathbb{F}_p[X]/(f(X) \mod p)$.

We will study in Chapters 1 to 6 error correcting codes over the finite rings

$$\frac{\mathbb{F}_q[[t]]/(t^r)[X]}{(f(X) \mod t^r)} \text{ and } \frac{\mathbb{Z}/p^r\mathbb{Z}[X]}{(f(X) \mod p^r)}.$$

They will often be denoted by B while A will designate the discrete valuation ring whose quotient by a power of its maximal ideal gives B. It would be easier to consider only the ring B as we study Reed-Solomon codes over B. But the proofs need a "division by the uniformizing parameter π of B". This is not possible due to the nilpotency of $\mathfrak{m}$. Therefore we use the classical strategy of "lifting". Instead of manipulating an element $x \in B$, we manipulate one of its liftings (a representant of x) in A where division by π is possible.

Galois rings that will be defined later form a subfamily of the quotients of DVRs. Throughout this PhD thesis they will be defined either as a quotient of $\mathbb{Z}_{p^d}$ or as an extension of degree d of $\mathbb{Z}/p^r\mathbb{Z}$. Both definition coincide by Theorem 21.

In chapter 2 we will consider more general rings even though we present timings only for discrete valuation rings. The considered rings are local domains but their maximal ideal is generated by more than one elements. The reader can safely read the chapter thinking we manipulate DVRs.

In addition, we study Reed-Solomon codes over non commutative rings in Chapter 3. Again in this case we consider the particular case of non commutative rings which are similar to discrete valuation rings. All the necessary materials on non commutative rings is written when necessary in the chapter. The needed results can be found in [MRS01].

Let now A be a ring. As in the commutative case we will consider modules over A. But one has to be careful for the scalar multiplication. Let $a_1, \ldots, a_m \in A^n$. Then we have in general

$$\sum_{i=1}^{m} Aa_i \neq \sum_{i=1}^{m} a_i A$$

We have to consider *left* and *right* modules over A.

Definition 23. Let A be a ring. A *left (resp. right) A-module* M is a commutative group together with an operation $A \times M \to M$ (resp. $M \times A \to M$) such that for all $a, b \in A$ and $x, y \in M$

- $a(x+y) = ax + ay$ (resp. $(x+y)a = xa + ya$),
- $(a+b)x = ax + bx$ (resp. $x(a+b) = xa + xb$),
- $a(bx) = (ab)x$ (resp. $(xa)b = x(ab)$) and
- $1x = x$ (resp. $x1 = x$).

We denote by $Z(A)$ the subring of A consisting of the elements which commutes with all the elements of A

$$Z(A) := \{a \in A : \forall b \in A \quad ab = ba\}.$$

Note that if the components of $a_1, \ldots, a_m$ are in the center of A, then we have $A(a_1, \ldots, a_m) = (a_1, \ldots, a_m)A$.

Complexity model

In order to analyze the performances of our algorithms, we let $\mathsf{I}(n)$ be the time needed to multiply two integers of bit-size at most n in *binary representation*. It is classical [CK91, Für07, SS71] that we can take $\mathsf{I}(n) \in O(n \log n 2^{\log^* n})$, where $\log^*$ represents the iterated logarithm of n. If A is a commutative ring, we let $\mathsf{M}_A(n)$ be the cost of multiplying two polynomials of degree at most n with coefficients in A in terms of the number of arithmetic operations in A. It is well known [GG03, Theorem 8.23, page 240] that we can take $\mathsf{M}_A(n) \in \tilde{O}(n)$. Thus the bit-cost of multiplying two elements of $\mathbb{F}_{p^n}$ is $\tilde{O}(n \log p)$ where p is a prime number.

Finally, let us recall that the *expected cost* spent by a randomized algorithm is defined as the average cost for a given input over all the possible executions.

Motivations and contributions of this PhD thesis

So far, we have only considered Reed-Solomon or generalized Reed-Solomon codes over fields and particularly finite fields. They can also be constructed over any ring (not necessarily commutative) with identity. They were studied over Galois rings in [Arm05b].

Proposition 24. *Let p be a prime and r, s two positive integers. Let $\varphi(X), \phi(X) \in \mathbb{Z}/p^r\mathbb{Z}[X]$ be two degree-s monic polynomials irreducible modulo p. Then there is a ring isomorphism*

$$\frac{\mathbb{Z}/p^r\mathbb{Z}[X]}{(\varphi(X))} = \frac{\mathbb{Z}/p^r\mathbb{Z}[X]}{(\phi(X))}.$$

The proof can be found in [Rag69, Statements I and II, page 207].

Definition 25 (Galois rings)**.** The ring $\frac{\mathbb{Z}/p^r\mathbb{Z}[X]}{(\varphi(X))}$ from the previous proposition is denoted by $\mathrm{GR}(p^r, s)$ and called a *Galois ring*.

The name for Galois rings comes from the following fact whose proof can also be found in [Rag69, Proposition 2, page 213].

Proposition 26. *Taking the same notation as the above definition, the ring automorphisms of* $\mathrm{GR}(p^r, s)$ *form a cyclic group of order* s*. (It is isomorphic to* $\mathrm{Gal}(\mathbb{F}_{p^{rs}}|\mathbb{F}_{p^r})$*.)*

The decoding of generalized Reed-Solomon codes over Galois rings have been widely studied. The unique decoding is treated in [IPE97, Nor99, NSM00, BF01, BF02, Arm02, Arm05c] while the list decoding is investigated in [Arm04, Arm05b, Arm05a, AdT05]. Generalized Reed-Solomon codes over commutative rings with identity are defined as in the field case but with extra condition on their support. For any ring A we denote by $A^\times$ the group of units of A.

Definition 27. Let A be a commutative ring with identity, $k < n$ be two positive integers, $x = (x_1, \ldots, x_n) \in A^n$ be such that for all $i \neq j$, $x_i - x_j \in A^\times$. The submodule generated by the

$$(f(x_1), \ldots, f(x_n)) \in A^n, f \in A[X] \text{ and } \deg f < k$$

is called a *Reed-Solomon code of parameters* $[n, k]_A$ and denoted by $\mathrm{RS}_A(x, n)$ or simply $\mathrm{RS}(n, k)$ when there is no confusion. The vector x is called the *support* of $\mathrm{RS}_A(x, k)$.

The condition "$i \neq j \Rightarrow x_i - x_j \in A^\times$" has several important consequences for Reed-Solomon codes over commutative rings with identity:

- The minimal distance of $\mathrm{RS}_A(n, k)$ is $n - k + 1$.
- The Welch-Berlekamp algorithm works *as is*, the proof of the algorithm need not be changed when the interpolation step is done with linear algebra.
- The Guruswami-Sudan algorithm works *as is*, the proof of the algorithm need not be changed when the interpolation step is done with linear algebra.
- Other techniques, that will be presented later, for decoding also work.

However this condition has a serious drawback for finite commutative rings with identity. Let A be such a ring, then by [AM94, Theorem 8.7, page 90]

$$A = \prod_{i=1}^{r} \mathfrak{A}_i$$

where $\mathfrak{A}_i$ is a finite local commutative ring with identity for $i = 1, \ldots, r$. Thus, if X designates the elements from A of the support of a Reed-Solomon code then

$$|X| \leq \min_{i \in \{1, \ldots, r\}} |\mathfrak{A}_i / \mathfrak{m}_i| .$$

where $\mathfrak{m}_i$ is the maximal ideal of $\mathfrak{A}_i$ for $i = 1, \ldots, n$. To fix the ideas, suppose that $A = \mathbb{Z}/p^r\mathbb{Z}$. Then $n \leq p \ll p^r = |A|$.

The motivations for studying codes over rings

Before giving the contributions, we explain in this subsection the motivations for the study of error correcting codes over rings.

The first two parts of this document describe two different approaches to solve the same problem: decoding Reed-Solomon codes over discrete valuation rings. Chapters 1 and 2 adapt the classical algorithm of Guruswami-Sudan for fields to DVRs while Chapters 3 and 4 present and study the properties of a lifting technique. The main motivation for the study of both approaches is to compare them from a theoretical point of view and then compare their implementations.

As stated in the previous section, Reed-Solomon codes over rings have a serious drawback, their length is small compared to the cardinality of the ring $n = o(|A|)$. Therefore they are not suitable for applications that requires $|A| = O(n)$.

However they can be used, for example, in the context of private information retrieval (PIR) [CGKS95]. PIR is a way of fetching an item from a database server without the server learning which item you are interested in. In [Gol07a] the author mentions the possibility of using finite commutative rings instead of finite fields for the alphabet of the underlying Reed-Solomon code. The author studies the special case of the rings $\mathbb{Z}/pq\mathbb{Z}$ where p and q are two distinct primes. Reed-Solomon codes over rings can also be used along with Shamir's secret sharing [Sha79, MS81] when the size of the secret has to be large in comparison to number of pieces. In both situations Reed-Solomon codes over quotients of discrete valuation rings have the advantage to have more efficient algorithms than their counterparts over finite fields of the same sizes. This is shown in Chapter 3.

Other motivations for the theoretical study of codes over rings are detailed in Chapter 4 and Chapter 5.

In Chapter 4, it is proved that interleaved codes over a finite field can be seen as codes over a finite ring, namely quotients of truncated power series rings $\kappa[[t]]$. Therefore any decoding algorithm for Reed-Solomon codes over $\kappa[[t]]/(t^r)$ induce a decoding algorithm for interleaved Reed-Solomon codes over κ.

In Chapter 5 we study quasi-cyclic codes and show that, as for cyclic codes, quasi-cyclic codes can be viewed as left ideal of a polynomial ring with square matrices coefficients. Given a square matrix $\Gamma \in M_{3\times 3}(\mathbb{F}_4)$ which is a primitive root of unity, we construct a Reed-Solomon code whose support is constituted by the powers of Γ. Then applying a projection we obtain a good code over $\mathbb{F}_4$, beating minimal distances with a fixed length and dimension. This suggest to further investigate Reed-Solomon codes over the rings of square matrices and, in fact, over an arbitrary (not necessarily commutative) ring, as it is cheap. A subfamily of quasi-cyclic codes comes from Reed-Solomon over a square matrix ring. Let A be any commutative ring and consider a primitive m-th root of unity $\Gamma \in M_{\ell\times\ell}(A)$. It is easy to see that the left submodule spanned by the vectors

$$\big(f(\Gamma^0), f(\Gamma^1), f(\Gamma^2), \ldots, f(\Gamma^{m-1})\big)$$

is a cyclic Reed-Solomon code over $M_{\ell\times\ell}(A)$. Then applying the projection

$$\begin{array}{rccc} \mathrm{pr}: & M_{\ell\times\ell}(A) & \longrightarrow & A^\ell \\ & \begin{pmatrix} a_{11} & \dots & a_{1\ell} \\ \vdots & & \vdots \\ a_{\ell 1} & \dots & a_{\ell\ell} \end{pmatrix} & \longmapsto & (a_{11},\dots,a_{1\ell}) \end{array}$$

to each components of the codewords, we obtain a quasi-cyclic code over A of length $m\ell$ and block size ℓ. Therefore, the given decoding algorithms of Chapter 3 for non commutative Reed-Solomon codes induce decoding algorithms for a subfamily of quasi-cyclic codes. This is a first step towards the decoding of quasi-cyclic codes needed, for example, in the context of McEliece cryptosystems [BCGO09].

Algebraic geometric codes have been considered over finite rings in [Bar06, WB08, VW99]. The Guruswami-Sudan decoding techniques have been applied to the latter but the interpolation and root-finding steps were left aside. As Reed-Solomon codes over rings form a special case of algebraic geometric codes over rings, Chapters 1 to 4 form a first step towards the decoding of the latter codes.

Chapter 6 gives the first decoding algorithm reaching the Johnson bound for number fields codes. Number fields codes are the analogue of Reed-Solomon over $\mathbb{Z}$ and $\mathcal{O}_K$. They can be used in the context of parallel computing. Suppose you want to compute the determinant of a big matrix with coefficients in $\mathcal{O}_K$, one can apply the Chinese remaindering theorem and distribute the computation among several computers. Each computer will work with a different prime ideal $\mathfrak{p} \subset \mathcal{O}_K$ on the matrix. When each computer has finished its computation the result can be built again thanks to the Chinese remaindering theorem. It can happens that a computer breaks or returns a wrong result. Therefore, the use of CRT codes can prevent the final result computation from failing.

Part I of the document: The Classical List Decoding Framework for Finite Rings

Although there have been several papers concerning the unique decoding of generalized Reed-Solomon codes, and more generally other families of codes over rings, no detailed complexities is given. The situation is the same for list decoding. Only one interpolation algorithm is given in [Arm05b] with no complexity. Moreover no root finding algorithm is given. In Part I we study, give algorithms and their complexities for list decoding over quotient of discrete valuation rings.

Definition 28. A *discrete valuation ring* (DVR) is a principal ideal domain which has one and only one prime ideal $\mathfrak{m}$. Any element $\pi \in A$ such that $(\pi) = \mathfrak{m}$ is called a *uniformizing parameter of* A. We say that $a \in A$ has valuation i and denote it by $v(a) = i$ if $a \in \mathfrak{m}^i \setminus \mathfrak{m}^{i+1}$. See [Ser62, Chapter 1] for more on DVRs.

First, in Chapter 1 we give an interpolation algorithm for generalized Reed-Solomon codes over Galois rings and its complexity. It follows the same idea as the interpolation algorithm of [Ale05].

Corollary 29. *For a Reed-Solomon code of parameters $[n,k]_B$, one can perform the interpolation step of the Sudan algorithm with a number of*

$$O\left(n^7k^2\left(\frac{n}{k}\right)^3\right)$$

arithmetic operations in B and

$$O\left(rn^6k^2\left(\frac{n}{k}\right)^3\right)$$

multiplications by π.

- *Or a number of*

$$\tilde{O}\left(n^7k^2\left(\frac{n}{k}\right)^3 rs\log p\right)$$

  bit-operations and

$$O\left(rn^6k^2\left(\frac{n}{k}\right)^3\right)$$

  multiplications by p when B is the Galois ring $\mathrm{GR}(p^r,s)$ and

- *Or a number of*

$$\tilde{O}\left(n^7k^2\left(\frac{n}{k}\right)^3 r\right)$$

  bit-operations and

$$O\left(rn^6k^2\left(\frac{n}{k}\right)^3\right)$$

  multiplications by t when B is the truncated power series ring $\mathbb{F}_q[[t]]/(t^r)$.

Then in Chapter 2 we give the first root finding algorithm for the second step of the Guruswami-Sudan algorithm for generalized Reed-Solomon codes over Galois rings and its complexity. We first study the structure of the set of roots of univariate polynomials.

Notation 30. Let R be a DVR. For any $i \in \mathbb{N}$, the subset of the elements of R of valuation at least i is written O_i.

Theorem 31. *Let R be a DVR with uniformizing parameter π. If F is a polynomial in $O_0[x]$ such that $F \mod \pi^n \neq 0$, then its set of roots in R to precision n can be written as the disjoint union of at most $\deg F$ classes of the form $a + O_i$.*

Then we give the first algorithm of root finding for univariate polynomials with coefficients in a certain finite commutative ring including Galois rings and truncated power series rings.

Theorem 32. *Let R be the power series ring $\mathbb{F}_q[[t]]$ over the finite field with $q = p^k$ elements. Then, for any polynomial F in $R[x]$ of degree at most d given to precision n, one can compute a set of at most d disjoint classes representing its set of roots in R to precision n with a randomized algorithm that performs an expected number of*

$$O\left((n\mathsf{M}(n) + \log(dq))\mathsf{M}(d)\log d + n\frac{d}{p}\log(q/p)\right)$$

operations in $\mathbb{F}_q$.

Theorem 33. *Let R be an unramified extension of $\mathbb{Z}_p$ of degree k. Then, for any given polynomial F in $R[x]$ of degree at most d given to precision n, one can compute a set of at most d disjoint classes representing its set of roots in R to precision n with a randomized algorithm that performs an expected number of $\tilde{O}((n + k\log p)ndk\log p)$ bit-operations.*

We then give an algorithm to find the set of roots of bivariate polynomials with coefficients in a Galois ring.

Theorem 34. *Let R be an unramified extension of $\mathbb{Z}_p$ of degree k and $F \in R[t][x]$ be such that $F \mod p^n \neq 0$. We denote by d the degree in x and by d_t the degree in t. Then one can compute the roots of F in $R[[t]]$ to precision l with an expected number of $\tilde{O}((n^2 + n(dl + d_t)k\log p)d(dl + d_t)k\log p)$ bit-operations.*

All the algorithms given in Chapter 2 have been implemented as we will see later in Part IV.

Part II of the document: A Lifting Framework for List Decoding over some Finite Rings

In this part we exploit the structure of some finite rings (such as the Galois rings) to obtain decoding algorithms over rings from decoding algorithms over finite fields. The first part of Chapter 3 is dedicated to generalized Reed-Solomon codes over non necessarily commutative rings with identity. We give their definition which is the same as generalized Reed-Solomon codes over commutative rings with identity with an extra condition.

Definition 35 (Generalized Reed-Solomon code over non commutative rings)**.** We fix three positive integers $k < n$ and $d = n-k+1$, a subset $\{x_1, \ldots, x_n\}$ of A such that for all $i \neq j$ $x_i - x_j \in A^\times$ and $x_i x_j = x_j x_i$, $x = (x_1, \ldots, x_n)$ and $v = (v_1, \ldots, v_n) \in (Z(A)^\times)^n$. The left submodule of A^n generated by the vectors of the form

$$(v_1 f(x_1), \ldots, v_n f(x_n)) \in A^n,$$

with $f \in A[X]_{<k}$ is denoted by

$$\mathrm{GRS}_A(v, x, k) = \mathrm{GRS}_A((v_1, \ldots, v_n), (x_1, \ldots, x_n), k)$$

and is called the *generalized Reed-Solomon* code over A of parameters $[v, x, k]$ or simply $[n, k]$ if there is no confusion on v and x. The integer n is called the *code block length* or simply *length* of $\mathrm{GRS}_A(v, x, k)$. The n-tuple $v = (v_1, \ldots, v_n)$ is called the *weight* of $\mathrm{GRS}_A(v, x, k)$. The n-tuple $x = (x_1, \ldots, x_n)$ is called the *support* of the code. When there is no confusion on the ring A, the weight and the support, we will simply write $\mathrm{GRS}(n, k)$ for $\mathrm{GRS}_A(v, x, k)$. The integer k will be called the *pseudo-dimension* of $\mathrm{GRS}(n, k)$ throughout this document. When $v = (1_A, \ldots, 1_A)$ we call $\mathrm{GRS}_A(v, x, k)$ a Reed-Solomon code and denote it by $\mathrm{RS}_A(v, x, k)$ or simply $\mathrm{RS}(n, k)$ if there is no confusion on the ring A, the weight and the support.

We prove that several properties about generalized Reed-Solomon codes over non commutative rings remain valid as well as decoding algorithms thanks to the simple following lemma:

Lemma 36. *Let $n < m$ be two positive integers and $M \in M_{n\times m}(A)$. Then there exists a nonzero $v \in A^m$ such that $Mv = 0$.*

Proof. The matrix M induces a group homomorphism of $A^m \to A^n$. Its kernel H is a subgroup of A^m of cardinality at least $|A|^{m-n} > 0$. □

Theorem 37. *A generalized Reed-Solomon codes over any ring A with identity of parameters $[n, k]_A$ is a free left submodule of A^n of dimension k with minimum distance $d = n - k + 1$. Moreover the Welch-Berlekamp algorithm can correct up to*

$$\left\lfloor \frac{d-1}{2} \right\rfloor = \left\lfloor \frac{n-k}{2} \right\rfloor$$

errors and, provided that the support of the code is contained in the center of A, the Guruswami-Sudan algorithm can correct up to

$$\left\lceil n - \sqrt{n(k-1)} \right\rceil - 1.$$

errors.

We also prove that generalized Reed-Solomon codes over non commutative rings are no better than their counterparts over commutative rings which themselves are no better than the counterparts over finite fields in the following sense:

Theorem 38. *Given three positive integers $k < n \leq q$, let A be a non commutative ring of cardinality q and a GRS code over A of parameters $[n, k, n - k + 1]_A$. Then there exists a commutative ring B of cardinality q and a GRS code over B of parameters $[n, k, n - k + 1]_B$.*

The same theorem holds when q is a prime power and if we replace "noncommutative rings" by "commutative rings" and "commutative rings" by "finite fields".

In the second part of Chapter 3 we generalize the lifting technique given in [GV98] to obtain decoding algorithms for generalized Reed-Solomon over non commutative rings and show several of their properties. Let A be a ring with identity with the following property:

(∗) *there exists a regular element p which is not a unit such that $p \in Z(A)$ and such that every element $a \in A$ can be uniquely written as $\sum_{i=0}^{\infty} a_i p^i$ where, for all $i \in \mathbb{N}$, a_i is in a set of representatives of $A/(p)$.*

For example A can be

- the power series ring over a field κ, $\kappa[[t]]$,
- an unramified extension of degree s of the p-adic ring $\mathbb{Z}_p$, $\mathbb{Z}_{p^s}$.
- the matrix ring over $\kappa[[t]]$, $M_\ell(\kappa[[t]])$,
- the matrix ring over $\mathbb{Z}_{p^s}$, $M_\ell(\mathbb{Z}_{p^s})$.

We study in more details the case of commutative rings. Suppose that A is commutative. Let r be a positive integer and $B = A/(p^r)$ and let $\mathcal{C}$ be a generalized Reed-Solomon code over B of parameters $[n,k]_B$. The idea behind the algorithms we present in this part is to take advantage of the decoding algorithms of generalized Reed-Solomon codes over a finite fields which have been widely studied in [Ber84, BW86, TERH88] for the unique decoding and in [Köt96, Sud97b, RR98, GS98, Ale05, AZ08] for the list decoding.

Proposition 39. *Given a Galois ring $A = \mathrm{GR}(p^r, s)$ and a RS code over A with parameters $[n, k, n-k+1]_A$, there exists a unique decoding with an asymptotic complexity of $\tilde{O}(rnks \log p)$ bit-operations; and a list decoding algorithm with an asymptotic complexity of $\tilde{O}(n^{r+6}k^5 s p^{rs(r-1)})$ bit-operations which can list decode up to the Johnson bound.*

The lifting algorithm has been first proposed in [GV98] then studied in [Byr01,BZ01]. Then in Chapter 4 we show how to improve, using erasures, the decoding radius of the lifting algorithm from Chapter 3. The idea of using erasures was first proposed by M. A. Armand in [Arm04] but he did not apply it within the lifting algorithm of M. Greferath and U. Vellbinger [GV98].

Notation 40. Let $B = A/(p^r)$ be such that there exists an integer q satisfying $q = \frac{A/p^r}{A/p^{r+1}}$ for all $r \in \mathbb{N}$, a map $\tau : \mathbb{N} \to \mathbb{N}$ and $\mathcal{C}$ be a code over B with parameters $[n,k]_B$. We let

$$N(\epsilon, \mathcal{C}, w) := \binom{n}{\epsilon} q^{r\epsilon} \binom{n-\epsilon}{w} \times \sum_{(v_0,\ldots,v_{r-1}) \in V_w} \left[\prod_{i=0}^{r-1} \binom{w - v_0 - \cdots - v_{i-1}}{v_i} (q-1)^{v_i} q^{v_0+\cdots+v_{i-1}} \right]$$

where

$$V_w = \{(v_0, \ldots, v_{r-1}) \in \mathbb{N}^r : v_0 + \cdots + v_{r-1} = w \text{ and } 0 \le v_0 \le \tau(\epsilon) \text{ and } 0 \le v_{i-1} \le \tau(\epsilon + v_0 + \cdots + v_{i-2}) \text{ for } i = 2, \ldots, r-1\},$$

and

$$P(\epsilon, \mathcal{C}, w) = \frac{\sum_{i=0}^{w} N(\epsilon, B, w)}{\sum_{i=0}^{w} \binom{n}{i} (q^r - 1)^i}$$

Theorem 41. *Taking the notation above with $A = \mathbb{Z}_{p^s}$ (an unramified extension of $\mathbb{Z}_p$ of degree s), let $\mathcal{C}$ be a Reed-Solomon code over $B = A/(p^r)$ with parameters $[n, k, d = n - k + 1]_B$ then there exists a unique decoding algorithm which can correct up to w errors and ϵ erasures with $w \leq n - \epsilon - k$ and which does succeed for a fraction of at least $P(\epsilon, B, w)$ error patterns with an expected number of $\tilde{O}(rnks \log p)$ bit-operations.*

We then show how to apply the improved lifting algorithms to interleaved linear codes.

Definition 42. We let A be the power series ring over the finite field $\mathbb{F}_q$ and $B = \mathbb{F}_q[[t]]/(t^r)$. We let $\mathcal{C}$ be a linear code over $\mathbb{F}_q$ with parameters $[n, k, d]_{\mathbb{F}_q}$ and with generator matrix G. Let r messages $m_0, \ldots, m_{r-1} \in \mathbb{F}_q^k$ and their encodings $c_0 = m_0 G, \ldots, c_{r-1} = m_{r-1} G$. For $i = 0, \ldots, r-1$ and $j = 1, \ldots, n$ define c_{ij} to be the j-th coordinate of c_i and $s_j = (c_{0,j}, \ldots, c_{r-1,j})$.

$c_{0,1}$	$c_{0,2}$	$\cdots$	$c_{0,n}$	$\rightarrow$	c_0
$c_{1,1}$	$c_{1,2}$	$\cdots$	$c_{1,n}$	$\rightarrow$	c_1
$\vdots$	$\vdots$	$\vdots$	$\vdots$		
$c_{r-1,1}$	$c_{r-1,2}$	$\cdots$	$c_{r-1,n}$	$\rightarrow$	c_{r-1}
$\downarrow$	$\downarrow$		$\downarrow$		
s_1	s_2		s_n		

The vectors transmitted over the channel are not $c_1, \ldots, c_{r-1} \in \mathbb{F}_q^n$ but $s_1, \ldots, s_n \in \mathbb{F}_q^r$. We will make an abuse of notation and call such an encoding scheme a *interleaved code with respect to $\mathcal{C}$ and of degree r*. In this context a *burst error* is a set of errors occurring in only one column s_i for one index $i \in \{1, \ldots, n\}$.

Interleaved linear codes over finite fields are presented in [VVO89, Chapter 7, Section 5] and their decoding is studied in [BKY03, CS03, GGR11]. They have also been considered over Galois rings in [Arm10].

Theorem 43. *Given a linear code $\mathcal{C}'$ over $\mathbb{F}_q$ with parameters $[n, k, d]_{\mathbb{F}_q}$ and a unique decoding algorithm* `BlackBoxErasuresDec` *from errors and erasures that can correct ϵ erasures and $\tau(\epsilon)$ errors in $\mathrm{dec}(\mathcal{C}')$ arithmetic operations over $\mathbb{F}_q$, there exists a unique decoding algorithm for interleaved codes with respect to $\mathcal{C}'$ and of degree r from errors and erasures that can correct ϵ erasures and $\tau(\epsilon)$ errors with at most $r\,\mathrm{dec}(\mathcal{C}')$ arithmetic operations over $\mathbb{F}_q$. Moreover it can correct at least a fraction of $P(\epsilon, B, w)$ error patterns of Hamming weight at most $w > \tau(\epsilon)$ over B also with at most $r\,\mathrm{dec}(\mathcal{C}')$ arithmetic operations over $\mathbb{F}_q$.*

	2	3	4	5	6
7	1.0	1.0	1.0	1.0	1.0
8	0.96	0.98	0.99	0.99	0.99
9	0.81	0.94	0.96	0.97	0.98
10	0.49	0.80	0.88	0.91	0.91
11	0.0073	0.53	0.70	0.75	0.78
12	0.00012	0.14	0.38	0.48	0.53

Figure 1: Fraction of correctible error patterns for a Goppa code of parameters $[256, 200, 15]_{\mathbb{F}_2}$.

	3	4	5	6
22	1.00000	1.00000	1.00000	1.00000
23	0.999997	0.999999	0.999999	0.999999
25	0.999844	0.999963	0.999981	0.999987
27	0.998099	0.999469	0.999715	0.999789
28	0.995114	0.998531	0.999185	0.999391
29	0.989079	0.996477	0.997984	0.998470
30	0.978112	0.992458	0.995554	0.996581

Figure 2: Fraction of corrigible error patterns for an Extended BCH code with parameters $[256, 100, 46]_{\mathbb{F}_2}$.

In Tables 1 and 2, the first row gives the degrees of interleaving and the first column shows the number of errors up to which we want to decode. The second row corresponds to half the minimum distance and, as expected, all of the probabilities are 1.0. We can see that the fraction of corrigible error patterns increases with the degree of interleaving and that codes with a high minimal distance are good candidates for interleaving.

We provide some application of our algorithm to linear codes over $\mathbb{F}_2$ and show in some tables at the end of Chapter 4 that the fraction of error patterns that can be corrected is high.

Related work on error correcting codes

In Part III I present other results obtained during my PhD thesis. They concern quasi cyclic codes whose definition is given below and number fields codes which are introduced later in this subsection.

Quasi cyclic codes

Let $n = m\ell$ be three positive integers.

Definition 44. Let $\mathcal{C} \subseteq \mathbb{F}_q^n$ be a code. We say that $\mathcal{C}$ is *cyclic* if

$$(c_1, \ldots, c_{n-1}, c_n) \in \mathcal{C} \Rightarrow (c_n, c_1, \ldots, c_{n-1}) \in \mathcal{C}$$

and we say that $\mathcal{C}$ is *ℓ-quasi-cyclic* if

$$(c_1, \ldots, c_n) \in \mathcal{C} \Rightarrow (c_{n-\ell+1}, \ldots, c_n, c_1, \ldots, c_{n-\ell}) \in \mathcal{C}.$$

It is well known [MS86a, Theorem 1, page 190] that there is a one-to-one correspondence between cyclic codes and ideals of the ring $\mathbb{F}_q[X]/(X^n - 1)$. We show that this correspondence can be extended to quasi cyclic codes.

Theorem 1. *There is a one-to-one correspondence between ℓ-quasi-cyclic codes over $\mathbb{F}_q$ of length $m\ell$ and left ideals of $M_\ell(\mathbb{F}_q)[X]/(X^m - 1)$.*

We extend also the definition of BCH codes to this context and use them to find new good codes over $\mathbb{F}_4$, codes beating known minimum distances with a fixed length and dimension, with the help of Markus Grassl.

Definition 45 (Primitive root of unity)**.** Let q be a prime power. A matrix $A \in M_\ell(\mathbb{F}_{q^s})$ is called a *primitive m-th root of unity* if

- $A^m = I_\ell$,
- $A^i \neq I_\ell$ if $i < m$,
- $\det(A^i - A^j) \neq 0$, whenever $i \neq j$.

New codes over $\mathbb{F}_4$				
$[171, 11, 109]_4$	$[172, 11, 110]_4$	$[173, 11, 110]_4$	$[174, 11, 111]_4$	$[175, 11, 112]_4$
$[176, 11, 113]_4$	$[177, 11, 114]_4$	$[178, 11, 115]_4$	$[179, 11, 115]_4$	$[180, 11, 116]_4$
$[181, 11, 117]_4$	$[182, 11, 118]_4$	$[183, 11, 119]_4$	$[184, 10, 121]_4$	$[184, 11, 120]_4$
$[185, 10, 122]_4$	$[185, 11, 121]_4$	$[186, 10, 123]_4$	$[186, 11, 122]_4$	$[187, 10, 124]_4$
$[187, 11, 123]_4$	$[188, 10, 125]_4$	$[188, 11, 124]_4$	$[189, 10, 126]_4$	$[189, 11, 125]_4$
$[190, 10, 127]_4$	$[190, 11, 126]_4$	$[191, 10, 128]_4$	$[191, 11, 127]_4$	$[192, 11, 128]_4$
$[193, 11, 128]_4$	$[194, 11, 128]_4$	$[195, 11, 128]_4$	$[196, 11, 129]_4$	$[197, 11, 130]_4$
$[198, 11, 130]_4$	$[199, 11, 131]_4$	$[200, 11, 132]_4$	$[201, 10, 133]_4$	$[201, 11, 132]_4$
$[202, 10, 134]_4$	$[202, 11, 132]_4$	$[203, 10, 135]_4$	$[204, 10, 136]_4$	$[204, 11, 133]_4$
$[205, 11, 134]_4$	$[210, 11, 137]_4$	$[213, 11, 139]_4$	$[214, 11, 140]_4$	

Definition 46 (Left quasi-BCH codes)**.** Let A be a primitive m-th root of unity in $M_\ell(\mathbb{F}_{q^s})$ and $\delta \leq m$. We define the ℓ-quasi-BCH code of length $m\ell$, with respect to A, with designed minimum distance δ, over $\mathbb{F}_q$ by

$$\text{Q-BCH}_q(m, \ell, \delta, A) := \left\{ (c_1, \ldots, c_m) \in (\mathbb{F}_q^\ell)^m : \sum_{j=0}^{m-1} A^{ij} c_{j+1} = 0 \text{ for } i = 1, \ldots, \delta - 1 \right\}.$$

We call the linear map

$$\begin{array}{rcl} \mathcal{S}_A : (\mathbb{F}_q^\ell)^m & \to & (\mathbb{F}_{q^s}^\ell)^m \\ x = (x_1, \ldots, x_m) & \mapsto & \sum_{j=0}^{m-1} A^j x_{j+1} \end{array}$$

the *syndrome* map with respect to Q-BCH(m, ℓ, δ, A).

All our new good codes are available at `http://www.codetables.de/` [Gra07]. However the site does not reference correctly our codes. This is due to the huge amount of work that the maintainer of the site, Markus Grassl, has to do. All the codes are obtained from our $[189, 11, 125]_{\mathbb{F}_4}$-code. The site references it as `BarbierChabotQuintin`. For example, in the construction section of our $[188, 11, 124]_{\mathbb{F}_4}$-code on Grassl's website, it is written that is obtained by puncturing the $[188, 11, 125]_{\mathbb{F}_4}$-code, which is the well referenced-one.

Number fields codes

Number fields codes form a subfamily of *Chinese remaindering theorem*-codes (CRT codes) which were first studied in [GSS00] then in [Gur04, Chapter 7, page 147–175].

Definition 47. Let A be any commutative ring with identity and $I_1, \ldots, I_n$ be coprime ideals and let $E \subseteq A$. Then the CRT code denoted by $\text{CRT}((I_1, \ldots, I_n), E)$ is the set

$$\{(x \mod I_1, \ldots, x \mod I_n) : x \in E\}.$$

When $A = \mathbb{Z}$, $I_1 = (p_1), \ldots, I_n = (p_n)$ where $p_1, \ldots, p_n$ are primes and

$$E = \left\{ x \in \mathbb{Z} : 0 \leq x < \prod_{i=0}^{k} p_i \right\}$$

with $k < n$, we obtain a well studied class of codes [Man76,GRS99,Bon00,GSS00,Gur04]. This subfamily of CRT codes is *erroneously* called "CRT codes" in the literature. We will call them "CRT codes over $\mathbb{Z}$". We can extend the construction of CRT codes over $\mathbb{Z}$ to integer rings of number fields.

Definition 48. Let K be a finite extension of $\mathbb{Q}$. The *Hermitian norm* of an element $x \in K$ is

$$\|x\| := \sqrt{\sum_{i=1}^{[K:\mathbb{Q}]} |\sigma_i(x)|^2}$$

where the σ_i are the canonical embeddings $K \to \mathbb{C}$.

Definition 49. Let K be a finite extension of $\mathbb{Q}$ and $\mathcal{O}_K$ be the integral closure of $\mathbb{Z}$ in K. Let $\mathfrak{p}_1, \ldots, \mathfrak{p}_n$ be n integral prime ideals. The CRT code

$$\mathrm{CRT}\left((\mathfrak{p}_1, \ldots, \mathfrak{p}_n), \{x \in \mathcal{O}_K | \|x\| \leq B\}\right).$$

where B is a fixed integer is called a *number field* code.

Number fields codes have been studied in only two papers [Len86, Gur03]. Their list decoding has been quickly considered in [CH11]. The authors used the Coppersmith [Cop97] theorem to claim that they can list decode number fields codes. Unfortunately they did not give any algorithm, complexity or comparison of their decoding radius with the Johnson bound. It turns out that a direct application of their theorem shows that it does not reach the Johnson bound.

In Chapter 6 we give the first list decoding algorithm for number fields codes that can decode up to the Johnson bound.

Theorem 50. *Let K be a number fields of degree d and $\mathcal{O}_K$ its integer ring. Let $\varepsilon > 0$, $k < n$ and prime ideals $\mathfrak{p}_1, \cdots \mathfrak{p}_n$ satisfying $\mathcal{N}(\mathfrak{p}_i) < \mathcal{N}(\mathfrak{p}_{i+1})$ and $\log \mathcal{N}(\mathfrak{p}_{k+1}) \geq (k \log \mathcal{N}(\mathfrak{p}_k) + d^2)$, then with the previous notations, there exists a list decoding algorithm for*

$$\mathrm{CRT}\left((\mathfrak{p}_1, \ldots, \mathfrak{p}_n), \left\{ x \in \mathcal{O}_K : \|x\| \leq \prod_{i=1}^{n} \mathcal{N}(\mathfrak{p}_i)^{1/d} \right\}\right).$$

with decoding radius $n - \sqrt{k(n+\varepsilon)}$.

Part IV of the document: Implementation

In this part I present the implementation in C/C++ of some algorithms presented in the first two parts of the PhD thesis. First in Chapter 7, I present the C++ computer algebra system `Mathemagix` where I have implemented finite fields arithmetic with a special emphasis for finite fields of characteristic 2 in the `finitefiedz` package. I also present the implementation if the arithmetic of Galois rings and the root finding of univariate and bivariate polynomials over Galois rings and truncated power series rings in the `quintix` package. Then in Chapter 8, I present the implementation in C of list decoding algorithms in an independent library called `decoding`. This library allows one to have list decoding algorithms without having to install a big computer algebra system. To my knowledge there was no open source implementation available for list decoding generalized Reed-Solomon codes thus it was necessary to have a working implementation.

Part I

The Classical List Decoding Framework for Finite Rings

Context

In this part we extend the Guruswami-Sudan algorithm to generalized Reed-Solomon codes over finite rings with identity and we study its complexity. Let A be a finite ring with identity, A need not be commutative. We let $A^\times$ denote the units of A and $Z(A)$ denote the center of A, all the $a \in A$ such that for all $b \in A$, $ab = ba$. The construction of generalized Reed-Solomon codes over finite rings is a bit different than generalized Reed-Solomon codes over fields, in order to obtain maximum distance separable codes we must add two conditions on their support.

Definition. Let $k < n$ be two positive integers and $x = (x_1, \ldots, x_n) \in A^n$ be such that for all $i \neq j$, $x_i - x_j \in A^\times$ (†) and $x_i x_j = x_j x_i$ (††). The *left submodule* generated by the

$$(f(x_1), \ldots, f(x_n)) \in A^n, f \in A[X] \text{ and } \deg f < k$$

is called a *Reed-Solomon code of parameters* $[n, k]_A$ and denoted by $\mathrm{RS}_A(x, n)$ or simply $\mathrm{RS}(n, k)$ when there is no confusion. The vector x is called the *support* of $\mathrm{RS}_A(x, k)$.

Thanks to conditions (†) and (††) the Reed-Solomon code of parameters $[n, k]_A$ has minimum distance $n-k+1$. When A is commutative condition (††) is useless and when A is a finite field we find the classical definition of Reed-Solomon codes (Definition 8).

Definition. Let $k < n$ be two positive integers, $v = (v_1, \ldots, v_n)$ be such that $v_i \in Z(A)^\times$ for all i and $x = (x_1, \ldots, x_n) \in A^n$ be such that for all $i \neq j$, $x_i - x_j \in A^\times$ and $x_i x_j = x_j x_i$. The *left submodule* generated by the

$$(v_1 f(x_1), \ldots, v_n f(x_n)) \in A^n, f \in A[X] \text{ and } \deg f < k$$

is called a *generalized Reed-Solomon code of parameters* $[n, k]_A$ and denoted by $\mathrm{GRS}_A(v, x, n)$ or simply $\mathrm{GRS}(n, k)$ when there is no confusion. The vector x is called the *support* while v is called the *weight* of $\mathrm{GRS}_A(v, x, k)$.

As for Reed-Solomon codes the minimum distance of $\mathrm{GRS}(n, k)$ is $n - k + 1$. Generalized Reed-Solomon and Reed-Solomon codes over the ring of matrices over a finite field are considered for example in [BCQ12]. They have been studied when the base ring is commutative in [Arm04, Arm05b]. In this PhD thesis we focus on a particular family of rings called *Galois rings*.

Proposition. *Let p be a prime and r, s two positive integers. Let $\varphi(X), \phi(X) \in \mathbb{Z}/p^r\mathbb{Z}[X]$ be two degree-s monic polynomials irreducible modulo p. Then there is a ring isomorphism*

$$\frac{\mathbb{Z}/p^r\mathbb{Z}[X]}{(\varphi(X))} = \frac{\mathbb{Z}/p^r\mathbb{Z}[X]}{(\phi(X))}.$$

The proof can be found in [Rag69, Statements I and II, page 207].

Definition (Galois rings)**.** The ring $\frac{\mathbb{Z}/p^r\mathbb{Z}[X]}{(\varphi(X))}$ from the previous proposition is denoted by $\mathrm{GR}(p^r, s)$ and called a *Galois ring*.

The name for Galois rings comes from the following fact whose proof can also be found in [Rag69, Proposition 2, page 213].

Proposition. *Taking the same notation as the above definition, the ring automorphisms of* $\mathrm{GR}(p^r, s)$ *form a cyclic group of order* s*. (It is isomorphic to* $\mathrm{Gal}(\mathbb{F}_{p^{rs}}|\mathbb{F}_{p^r})$*.)*

The decoding of generalized Reed-Solomon codes over Galois rings have been widely studied. The unique decoding is treated in [IPE97, Nor99, NSM00, BF01, BF02, Arm02, Arm05c] while the list decoding is investigated in [Arm04, Arm05b, Arm05a, AdT05]. Algebraic geometric over local Artinian rings have also been studied [Wal99a, VW99, WB08] as well as their list decoding [Bar06, WB08].

In fact, when working with finite rings with identity as alphabets for generalized Reed-Solomon codes, their finiteness implies the following lemma which allows one to apply the Guruswami-Sudan algorithm over any finite ring as soon as one dispose of practical linear algebra algorithms over the latter.

Lemma. *Let* $n < m$ *be two positive integers and* $M \in M_{n\times m}(A)$*. Then there exists a nonzero* $v \in A^m$ *such that* $Mv = 0$*.*

Proof. The matrix M induces a group homomorphism of $A^m \to A^n$. Its kernel H is a subgroup of A^m of cardinality at least $|A|^{m-n} > 0$. □

When A is commutative, the previous lemma can also be proven with linear algebra techniques, see for example [Bro93, Chapter 5 and Corollary 5.9, page 39]. The Guruswami-Sudan algorithm for generalized Reed-Solomon codes over fields is constituted of two parts. Thanks to the previous lemma it can be applied *as is* to generalized Reed-Solomon codes over finite rings and commutative rings.

Algorithm 2 Overview of the Guruswami-Sudan algorithm.

Input: A received word $y \in A^n$ and a decoding radius τ.
Output: All codewords $c \in A^n$ such that $d(c, y) \leq \tau$.

1: With linear algebra, find a bivariate polynomial $Q(X, Y) \in A[X, Y]$ satisfying certain properties.
2: Find all the roots of $Q(X, Y)$ seen as a univariate polynomial of $(A[X])[Y]$.

The maximum decoding radius of the Algorithm can be computed in exactly the same way as in the case of finite fields. It will be shown in this thesis that

$$\left\lceil n - \sqrt{n(k-1)} \right\rceil - 1$$

is the maximum possible value for the Guruswami-Sudan algorithm, thus reaching the Johnson bound (Definition 15).

Contributions

Although there have been several papers concerning the unique decoding of generalized Reed-Solomon codes, and more generally other families of codes over rings, no detailed complexities is given. The situation is the same for list decoding. Only one interpolation algorithm is given in [Arm05b] with no complexity. Moreover no root finding algorithm is given. First, in Chapter 1 we give an interpolation algorithm for generalized Reed-Solomon codes over Galois rings and its complexity. It follows the same idea as the interpolation algorithm of [Ale05]. Then in Chapter 2 we give the first root finding algorithm for the second step of the Guruswami-Sudan algorithm for generalized Reed-Solomon codes over Galois rings and its complexity. We first give the first algorithm for root finding of polynomials with coefficients in a local finite commutative ring. We then give an algorithm to find the roots of bivariate polynomials with coefficients in a Galois ring.

Chapter 1

Shortest Vectors in Polynomial Lattices Over Galois Rings and Application to List Decoding

This chapter contains the most recent results I obtained. It has not been submitted. It concerns the interpolation step of the Guruswami-Sudan algorithm and is included here for the completeness of this part of the PhD thesis.

1.1 Introduction

In this chapter we adapt the algorithm of [Ale05], which works over finite fields, over discrete valuation rings. Given a finite field $\mathbb{F}_q$ and a lattice Λ in $\mathbb{F}_q[X]^n$, the algorithm in [Ale05] gives in polynomial time the shortest vector of Λ. The "norm" used here is the degree of the polynomial of greatest degree among all the components of a vector of Λ. It is interesting to note that the finding of the shortest vector of a lattice in $\mathbb{F}_q[X]^m$ can be done in polynomial time whereas, *a priori*, no polynomial time algorithm is known to find the shortest vector of a lattice in $\mathbb{Z}^m$.

We follow the presentation given in Alekhnovich's paper [Ale05]. We first recall the definition and properties of discrete valuation rings. Then we give a naive algorithm which computes the shortest vector of a lattice $\Lambda \subset (A/(\pi^r))^n$ where A is a DVR with uniformizing parameter π. Finally we apply this algorithm to the Sudan list decoding algorithm for Reed-Solomon codes over $A/(\pi^r)$.

1.1.1 Related work

Given a lattice in $\Lambda' \subset \mathbb{R}^m$ it is NP-hard [Ajt98] to find a shortest vector. However in the early 1980s a polynomial time algorithm for finding a short vector in Λ' was given in [LLL82]. The situation for polynomial lattices is much simpler. Let $\Lambda \subset \kappa[X]^m$ be a polynomial lattice over the field κ. One can compute a shortest vector of Λ in

polynomial time [MS03]. In [Ale05] the author applies a shortest vector finding algorithm to the Guruswami-Sudan problem. Related work of shortest vector in polynomial lattices includes [GJV03, JV05, SV05, JV06].

1.2 Prerequisites

1.2.1 Complexity model

The "soft-Oh" notation $f(n) \in \tilde{O}(g(n))$ means that $f(n) \in g(n)\log^{O(1)}(3+g(n))$. It is well known [Für07] that the time needed to multiply two integers of bit-size at most n in *binary representation* is $\tilde{O}(n)$. The cost of multiplying two polynomials of degree at most n over a ring A is $\tilde{O}(n)$ in terms of the number of arithmetic operations in A. Thus the bit-cost of multiplying two elements of the finite field $\mathbb{F}_{p^n}$ is $\tilde{O}(n\log p)$.

1.2.2 Discrete valuation rings

Definition 51. A *discrete valuation ring* (DVR) is a principal ideal domain which has one and only one prime ideal $\mathfrak{m}$. Any element $\pi \in A$ such that $(\pi) = \mathfrak{m}$ is called a *uniformizing parameter of* A. Let $a \in A$ be a nonzero element. The greatest integer $i \in \mathbb{N}$ such that $a \in \mathfrak{m}^i \setminus \mathfrak{m}^{i+1}$ is called the *valuation* of a and denoted by $v(a)$. We let $v(0) = +\infty$.

Definition 52. Let $\mathbb{Z}_{p^s}$ be an unramified extension of $\mathbb{Z}_p$ of degree s. Then $\mathbb{Z}_{p^s}$ is a DVR with uniformizing parameter p [Ser62, Chapter 1]. Let r be a positive integer. Then the quotient ring

$$\mathrm{GR}(p^r, s) := \frac{\mathbb{Z}_{p^s}}{(p^r)}$$

is called a *Galois ring*.

Proposition 53. *Let $a, b \in \mathrm{GR}(p^r, s)$. Then the product $ab \in \mathrm{GR}(p^r, s)$ can be computed with a number of*

$$O\left(s\log s\log\log sr\log p\log(r\log p)\log\log(r\log p)\right)$$

or $\tilde{O}(rs\log p)$ bit-operations.

Proposition 54. *Let $\mathbb{F}_q[[t]]$ be the power series ring over $\mathbb{F}_q$. It is a DVR with uniformizing parameter t. Let $a, b \in \mathbb{F}_q[[t]]/(t^r)$. Then the product $ab \in \mathbb{F}_q[[t]]/(t^r)$ can be computed with a number of $O(r\log r\log\log r)$ or $\tilde{O}(r)$ arithmetic operations in $\mathbb{F}_q$.*

Proof. Proposition 52 and 53 are in [GG03, Chapter 8]. □

1.2.3 Reed-Solomon codes over valuation rings

Let A be a DVR. Reed-Solomon codes over rings are defined in a slightly different way than their field counterparts. We let $A[X]_{<k}$ denote the submodule of $A[X]$ consisting of all the polynomials of degree at most $k-1$ of $A[X]$.

Definition 55. Let $x_1, \ldots, x_n$ be elements of A such that $x_i - x_j \in A^\times$ fir $i \neq j$ (where $A^\times$ is the group of units of A). The submodule of A^n generated by the vectors $(f(x_1), \ldots, f(x_n)) \in A^n$ where $f \in A[X]_{<k}$ is called a *Reed-Solomon* code over A. The n-tuple $(x_1, \ldots, x_n)$ is called the *support* of the RS code.

Proposition 56. *Let $\mathcal{C}$ be a RS code over A. Then $\mathcal{C}$ has parameters $[n, k, d = n-k+1]_A$.*

Proposition 57. *Let $\mathcal{C}$ be a RS code with parameters $[n, k, d = n-k+1]_A$ over a discrete valuation ring A with uniformizing parameter π. Then $\mathcal{C}/\pi^r\mathcal{C}$ is a RS code with parameters $[n, k, d]_{A/(\pi^r)}$ over $A/(\pi^r)$. Moreover of $(x_1, \ldots, x_n)$ is the support of $\mathcal{C}$ then $(x_1 \mod \pi^r, \ldots, x_n \mod \pi^r)$ is the support of $\mathcal{C}/\pi^r\mathcal{C}$.*

The decoding of Reed-Solomon codes over Galois rings have been widely studied. The unique decoding is treated in [IPE97, Nor99, NSM00, BF01, BF02, Arm02, Arm05c] while the list decoding is investigated in [Arm04, Arm05b, Arm05a, AdT05]. We let $\mathcal{C}$ be a Reed-Solomon code over A of parameters $[n, k]_A$, $J = \left\lceil n - \sqrt{n(k-1)} \right\rceil - 1$ and

$$S = n - (k-1)r_{kn} \left\lceil \frac{2n - (k-1)r_{kn}(r_{kn}+1)}{2(r_{kn}+1)} \right\rceil - 2 \text{ where } r_{kn} = \left\lfloor \sqrt{\frac{2(n+1)}{k-1} + \frac{1}{4}} - \frac{1}{2} \right\rfloor$$

We recall the Sudan and the Guruswami-Sudan algorithms.

Algorithm 3 Sudan

Input: a positive integer $\tau \leq S$ and a received vector y of A^n with at most τ errors.
Output: all the $f \in A[X]_{<k}$ such that $d(y, f(x)) \leq \tau$.

1: $L \leftarrow \left\lceil \frac{n-\tau-1}{k-1} \right\rceil - 1$.
2: Find $Q = \sum_{i=0}^{L} Q_i(X)Y^i \in (A[X])[Y]$ of degree at most L such that

   1. $Q(x_i, y_i) = 0$ for all $1 \leq i \leq n$.
   2. $\deg Q_i \leq (n-\tau) - 1 - i(k-1)$ for all $0 \leq i \leq L$.

3: $\mathcal{Z} \leftarrow$ Roots of Q in $A[X]_{<k}$ such that $d(y, f(x)) \leq \tau$.
4: **return** $\{(f(x_1), \ldots, f(x_n) : f \in \mathcal{Z}\}$

Proposition 58. *Algorithm 3 and 4 work correctly as expected.*

The proof of Proposition 58 follows exactly the same proof as in the finite field case. See for example [Sud97b] and [GS98]. The only point that differs concerns the existence of the polynomial $Q(X, Y)$ of step 3. But the following lemma allows us to use the same property as the one for matrices over commutative fields.

Algorithm 4 Guruswami-Sudan

Input: a positive integer $\tau \leq J$ and a received vector y of A^n with at most τ errors.
Output: all the $f \in A[X]_{<k}$ such that $d(y, f(x)) \leq \tau$.

1: $s \leftarrow \left\lfloor \frac{(k-1)n+\sqrt{(k-1)^2n^2+4((n-\tau)^2-(k-1)n)}}{2((n-\tau)^2-(k-1)n)} \right\rfloor + 1$.
2: $L \leftarrow \left\lceil \frac{s(n-\tau)-1}{k-1} \right\rceil - 1$.
3: Find $Q = \sum_{i=0}^{L} Q_i(X)Y^i \in (A[X])[Y]$ of degree at most L such that

1. $Q(x_i, y_i) = 0$ for all $1 \leq i \leq n$.
2. $Q(X + x_i, Y + y_i)$ has valuation at least s.
3. $\deg Q_i \leq s(n-\tau) - 1 - i(k-1)$ for all $0 \leq i \leq L$.

4: $\mathcal{Z} \leftarrow$ Roots of Q in $A[X]_{<k}$ such that $d(y, f(x)) \leq \tau$.
5: **return** $\{(f(x_1), \ldots, f(x_n) : f \in \mathcal{Z}\}$

Lemma 59. *Let $n < m$ be two positive integers and $M \in M_{n\times m}(A)$. Then there exists a nonzero $v \in A^m$ such that $Mv = 0$.*

Proof. The matrix M induces a group homomorphism of $A^m \to A^n$. Its kernel H is a subgroup of A^m of cardinality at least $|A|^{m-n} > 0$. It can also be proven with linear algebra techniques, see for example [Bro93, Chapter 5 and Corollary 5.9, page 39]. □

Informally speaking, note that the finding of Q (step 2 of Algorithm 3 and step 3 of Algorithm 4) can be seen as finding a vector of univariate polynomials with bounded degrees in a certain ideal.

1.3 Computing the shortest vector

1.3.1 Preliminaries

Until the end of this chapter, we let A be a DVR with uniformizing parameter π be such that $A/(\pi)$ is a finite field and $B = A/(\pi^r)$ for a positive integer r.

In this section we solve the following problem:

Problem 60. Let $M \in M_{m\times\ell}(B[X])$ be a matrix

$$\begin{pmatrix} M_{11}(X) & \ldots & M_{1\ell}(X) \\ \vdots & & \vdots \\ M_{m1}(X) & \ldots & M_{m\ell}(X) \end{pmatrix}$$

and let $f = (f_1(X), \ldots, f_\ell(X)) \in B[X]^\ell$. Let

$$\begin{aligned} d_1 &= \deg\left(f_1(X)M_{11}(X) + f_2(X)M_{12}(X) + \cdots + f_\ell M_{1\ell}(X)\right) \\ d_2 &= \deg\left(f_1(X)M_{21}(X) + f_2(X)M_{22}(X) + \cdots + f_\ell M_{2\ell}(X)\right) \\ &\cdots \\ d_m &= \deg\left(f_1(X)M_{m1}(X) + f_2(X)M_{m2}(X) + \cdots + f_\ell M_{m\ell}(X)\right). \end{aligned}$$

Our goal is to find $f \in B[X]^\ell$ such that $\max(d_1, d_2, \ldots, d_\ell)$ has the least possible nonzero value.

Problem 60 is equivalent to finding the shortest vector of the lattice

$$\Lambda = \left\langle \begin{pmatrix} M_{11}(X) \\ M_{21}(X) \\ \vdots \\ M_{m1}(X) \end{pmatrix}, \begin{pmatrix} M_{12}(X) \\ M_{22}(X) \\ \vdots \\ M_{m2}(X) \end{pmatrix}, \ldots, \begin{pmatrix} M_{1\ell}(X) \\ M_{2\ell}(X) \\ \vdots \\ M_{m\ell}(X) \end{pmatrix} \right\rangle.$$

where the "norm" of

$$g = (g_1(X), \ldots, g_\ell(X)) \in B[X]^\ell$$

is taken to be

$$\max\left(\deg g_1(X), \ldots, \deg g_\ell(X)\right).$$

In Subsection 1.4.1 we will present how to reduce the interpolation step of the Guruswami-Sudan algorithm to Problem 60 by a suitable choice of the matrix M.

1.3.2 The naive algorithm

We let M be the $B[X]$-module $(B[X])^m$ for a fixed integer m.

Definition 61. Let $b \in B$. Then there exists a unique unit $u \in B^\times$ and a non negative integer ν such that $b = u\pi^\nu$. The integer ν is called the *filtration* of b. The filtration of $0 \in B$ is defined to be $+\infty$. For an element $\varphi \in B[X]$ we define the *filtration* of φ to be the least filtration of the nonzero coefficients of φ. For an element $f = (f_1, \ldots, f_m) \in M$ the *filtration* of f is defined to be the least filtration of the components of f.

Definition 62. For any element $f = (f_1(X), \ldots, f_m(X)) \in M$, we define its *degree* $\deg f$ to be $\max\{\deg f_i : i \in \{1, \ldots, m\}\}$, its *leading index* $\mathrm{LI}(f)$ to be

$$\max\{i : i \in \{1, \ldots, m\} \text{ and } \deg f_i = \deg f\}$$

and its *leading coefficient* $\mathrm{LC}(f)$ to be $f_{\mathrm{LI}(f)}$.

The degree of $0 \in M$ is $-\infty$. If E is any subset of M non reduced to zero, we call a *vector of minimal degree* or *shortest vector of* E any *nonzero* vector a_0 satisfying $\forall b \in E, b \neq 0, \deg b \geq \deg a_0$.

Example 63. We let $B = \mathbb{Z}/4\mathbb{Z}$, then we have

$$\mathrm{LI}\left(\begin{pmatrix} X^2 \\ 1 \\ 2X^2 \\ 3X \end{pmatrix}\right) = 3$$

and

$$\mathrm{LC}\left(\begin{pmatrix} X^2 \\ 1 \\ 2X^2 \\ 3X \end{pmatrix}\right) = 2X^2.$$

If we consider the submodule N of $B[X]^2$ generated by the columns of

$$\begin{pmatrix} X^2 & 2X \\ 1 & 1 \end{pmatrix}$$

then a shortest vector of N is

$$\begin{pmatrix} 0 \\ 2 \end{pmatrix} = 2 \cdot \begin{pmatrix} 2X \\ 1 \end{pmatrix}$$

Definition 64. Let $a \in M$ and $s \in \mathbb{N}$ be such that $\pi^s a \neq 0$ and $\pi^{s+1} a = 0$. The vector $\pi^s a$ is denoted by a_π.

Definition 65. Let $\ell \leq m$ and $f = (f_1, \ldots, f_\ell) \in M^\ell$. We say that $f_1, \ldots, f_\ell$ are *π-independent* if there exists a $\ell \times \ell$ minor which is not divisible by π. We define

$$f_\pi := ((f_1)_\pi, \ldots, (f_\ell)_\pi).$$

Proposition 66. *Let $\ell \leq m$ and $f = (f_1, \ldots, f_\ell) \in M^\ell$ be π-independent vectors. Then $\langle f_\pi \rangle$ contains a shortest vector of $\langle f \rangle$.*

Proof. Let ψ be a shortest vector of $\langle f \rangle$ such that $\pi\psi = 0$. Such a ψ always exists and we can write

$$\psi = \sum_{i=1}^{\ell} a_i f_i$$

with $a_i \in B[X]$ for $i = 1, \ldots, \ell$. We have

$$\sum_{i=1}^{\ell} \pi a_i f_i = 0.$$

Let $P(X) \in B[X]$ be a monic polynomial irreducible modulo π of degree greater than $\max\{(\deg a_i + \deg f_i) : i \in \{1, \ldots, \ell\}\} + 1$. Then we have a homomorphism of $B[X]$-module

$$\begin{array}{ccc} M & \longrightarrow & \left(\frac{B[X]}{(P(X))}\right)^m \\ a & \longmapsto & \bar{a}. \end{array}$$

As the components of f are π-independent we have that $\bar{f} = (\bar{f}_1, \ldots, \bar{f}_\ell)$ are linearly independent over the ring $\frac{B[X]}{(P(X))}$ and we obtain that $\pi\bar{a}_i = 0$ which implies that π^{r-1} divides $\bar{a}_i$ for $i = 1, \ldots, \ell$. As $\deg a_i + \deg f_i < \deg P$ we also have that π^{r-1} divides a_i for $i = 1, \ldots, \ell$. □

Definition 67. For any finite set of vectors $E \subseteq M$, we define the *degree* of E, denoted by $\deg E$, to be $\sum_{f \in E} \deg f$ and the *max-degree*, denoted by $\operatorname{maxdeg} E$ to be $\max\{\deg f : f \in E\}$.

Definition 68. Let $a \in M$. We say that a is π-*reduced* if for any non negative integer v we have either $\pi^v a = 0$ or, $\mathrm{LI}(\pi^v a) = \mathrm{LI}(a)$ and $\deg(\pi^v a) = \deg a$. Let E be a finite set of vectors of M. We say that E is X-*reduced* if for all $a, b \in E$, $a \neq b$, we have $\mathrm{LI}(a) \neq \mathrm{LI}(b)$. Finally, we say that E is *reduced* if E is X-reduced and each element of E is π-reduced.

Example 69. Let B be the ring $\mathbb{Z}/2^2\mathbb{Z} = \mathbb{Z}_2/(2^2)$. The vector $(1, X)$ is 2-reduced while the vector $(1, 2X)$ is not as we have $2 \cdot (1, 2X) = (2, 0)$. The set

$$\left\{ \begin{pmatrix} 1 \\ X \end{pmatrix}, \begin{pmatrix} 2X \\ 1 \end{pmatrix} \right\}$$

is X-reduced while

$$\left\{ \begin{pmatrix} 1 \\ X \end{pmatrix}, \begin{pmatrix} 1 \\ 2X \end{pmatrix} \right\}$$

is not as we have

$$\mathrm{LI}\left(\begin{pmatrix} 1 \\ X \end{pmatrix} \right) = 2 = \mathrm{LI}\left(\begin{pmatrix} 1 \\ 2X \end{pmatrix} \right).$$

Lemma 70. *Let $a \in M$ be a nonzero π-reduced vector. Then for all $\lambda \in B[X]$ such that $\lambda a \neq 0$ we have $\mathrm{LI}(\lambda a) = \mathrm{LI}(a)$ and $\deg(\lambda a) = \deg \lambda + \deg a$.*

Proof. Let $d = \deg \lambda$ and $\lambda_d X^d$ be the leading term of λ. Write $a = (a_1(X), \ldots, a_m(X))$ and let $\alpha = a_{\mathrm{LI}(a)} \neq 0$. For $i = 1, \ldots, m$ we have $\deg \lambda a_i \leq \deg \lambda + \deg a_i$. Write $\lambda_d = u\pi^v$ where u is a unit of B and v is a non negative integer. Then, by hypothesis, we have $\deg a = \deg \alpha = \deg \pi^v \alpha = \deg \lambda_d \alpha$ hence the leading terms of α and $\lambda_d \alpha$ have the same degree. Therefore $\deg \lambda\alpha = \deg \lambda + \deg \alpha = \deg \lambda + \deg a$. Moreover for $i = 1, \ldots, m$ we have $\deg a_i \leq \deg \alpha$, thus $\deg \lambda a_i \leq \deg \lambda + \deg \alpha = \deg \lambda\alpha$ and $\deg \lambda a = \deg \lambda\alpha$. □

Proposition 71. *Let $\ell \leq m$ and $f = (f_1, \ldots, f_\ell)$ be reduced vectors of M. Then f contains a minimal vector of $\langle f \rangle$.*

Proof. Suppose that there exists $\lambda_1, \ldots, \lambda_\ell \in B[X]$ such that

$$\deg \left(\sum_{i=1}^{\ell} \lambda_i f_i \right) < \min\{\deg f_i : i \in \{1, \ldots, \ell\}\}. \tag{1.1}$$

Note that if $\lambda_i f_i \neq 0$ we have $\mathrm{LI}(\lambda_i f_i) = \mathrm{LI}(\lambda_i)$ and $\deg(\lambda_i f_i) = \deg \lambda_i + \deg f_i$ by Lemma 70. Among all the nonzero terms of the sum of (1.1) take the one of maximal degree and of maximal leading index, say, $\lambda_{i_0} f_{i_0}$. The leading coefficient of $\lambda_{i_0} f_{i_0}$ can only be canceled by a leading coefficient of another term which is impossible because f is X-reduced, hence a contradiction. □

We present an algorithm 5 which, given a set of vectors, compute X-reduced vectors.

Algorithm 5 X-reduction.

Input: $f = (f_1, \ldots, f_\ell) \neq 0$ with $\ell \leq m$.
Output: $f' = (f'_1, \ldots, f'_\ell)$ such that $0 \neq \langle f_\pi \rangle \subseteq \langle f' \rangle \subseteq \langle f \rangle$ and f' is X-reduced or $\deg f' < \deg f$. (It can happen that $f'_i = 0$ for some $i \in \{1, \ldots, \ell\}$.)

1: $f' \leftarrow f$.
2: **while** f' is not X-reduced or $\deg f' = \deg f$ **do**
3: Find $a \neq b$ in f' such that $\mathrm{LI}(a) = \mathrm{LI}(b)$ and $\deg a \geq \deg b$.
4: $i \leftarrow \mathrm{LI}(a) = \mathrm{LI}(b)$.
5: $u_a \pi^{v_a} X^{d_a} \leftarrow$ the leading term of a_i such that u_a is a unit of B.
6: $u_b \pi^{v_b} X^{d_b} \leftarrow$ the leading term of b_i such that u_b is a unit of B.
7: $s_a \leftarrow \max(0, v_b - v_a)$.
8: $s_b \leftarrow \max(0, v_a - v_b)$.
9: $a \leftarrow \pi^{s_a} u_b a - u_a \pi^{s_b} X^{d_a - d_b} b$.
10: **end while**
11: **return** f'.

Proposition 72. *Algorithm 5 works correctly and takes at most $O(m^2 \ell \operatorname{maxdeg} f)$ arithmetic operations in B.*

Proof. At step 2, if f is X-reduced, then the proposition holds. If f' is not X-reduced then we can always find $a, b \in f'$, $a \neq b$, $\mathrm{LI}(a) = \mathrm{LI}(b)$ and $\deg a \geq \deg b$ at step 3. At each iteration, for $i \in \{1, \ldots, \ell\}$ either

- $\mathrm{LI}(f_i)$ decreases and $\deg f = \deg f'$ which can happen at most m times, or
- $\mathrm{LI}(f_i)$ decreases and $\deg f < \deg f'$ in which case the algorithm finishes, or
- $\mathrm{LI}(f_i)$ does not change or increases and then $\deg f' < \deg f$ in which case the algorithm finishes.

Therefore the loop of Algorithm 5 from step 2 to 10 executes a maximum of ℓm iterations. At each iteration at most one vector of f' becomes zero. Suppose that we end up with only one element in f'. This means that f' is X-reduced and the algorithm finishes with $f' \neq 0$.

Each execution of step 9 takes at most $O(m \operatorname{maxdeg} f)$ arithmetic operations in B. Step 9 is executed at most $O(m^2 \ell \operatorname{maxdeg} f)$ times. □

We now present Algorithm 6 which, given a set of vectors, compute their π-reduction.

Algorithm 6 π-reduction.

Input: $f = (f_1, \ldots, f_\ell)$ with $\ell \leq m$.

Output: $f' = (f'_1, \ldots, f'_\ell)$ such that $0 \neq \langle f_\pi \rangle \subseteq \langle f' \rangle \subseteq \langle f \rangle$. and $0 \neq f'_i$ is π-reduced for $i = 1, \ldots, \ell$.

1: $f' \leftarrow f$.
2: **for** $i = 1 \to \ell$ **do**
3: **while** f_i is not π-reduced **do**
4: $f_i \leftarrow \pi f_i$.
5: **end while**
6: **end for**
7: **return** f'.

Proposition 73. *Algorithm 6 works correctly and requires at most* $O(rm\ell \operatorname{maxdeg} f)$ *multiplications by* π *in* B.

Proof. Let $f \in M$ be a non π-reduced vector. Let μ be the greatest integer such that $\pi^\mu f \neq 0$. Then $\pi^\mu f$ is π-reduced. Hence the inner loop (from step 3 to step 5) terminates and produce a nonzero π-reduced vector. The correctness of the algorithm follows. □

The algorithm that computes a reduced set f' from a finite set f of vectors of M is a simple loop that call successively call Algorithm 5 and 6 until we obtain a reduced set.

Algorithm 7 Reduction.

Input: $f = (f_1, \ldots, f_\ell)$ be π-independent vectors with $\ell \leq m$.

Output: $f' = (f'_1, \ldots, f'_\ell)$ such that $0 \neq \langle f_\pi \rangle \subseteq \langle f' \rangle \subseteq \langle f \rangle$. and f' is reduced. (It can happen that $f'_i = 0$ for some $i \in \{1, \ldots, \ell\}$.)

1: $f' \leftarrow f$.
2: **while** f' is not reduced **do**
3: **while** f' is not X-reduced **do**
4: Call algorithm 5 with input f' to obtain f'_1
5: $f' \leftarrow f'_1$.
6: **end while**
7: Call algorithm 6 with input f' to obtain f'_1.
8: $f' \leftarrow f'_1$.
9: **end while**
10: **return** f'.

Proposition 74. *Algorithm 7 works correctly and performs at most* $O(m^3\ell^2 \deg f \operatorname{maxdeg} f)$ *arithmetic operations in* B *and at most* $O(rm^2\ell^2 \deg f \operatorname{maxdeg} f)$ *multiplications by* π *in* B.

Proof. Algorithm 5 returns either a reduced set or a set with degree less than $\deg f'$ by Proposition 72.

Suppose that for each iteration of the loop from step 2 to 9, we have

- at step 4, $\deg f_1' = \deg f'$ and f_1' is not π-reduced and
- at step 7, $\deg f_1' = \deg f'$.

This implies for both step 4 and 7 that for each $i = 1, \ldots, \ell$ such that $(f_1')_i \neq 0$, $\mathrm{LI}((f_1')_i) < \mathrm{LI}(f_i')$. This can happen at most $m\ell$ times. Also it can happen at most $\deg f$ times that $\deg f_1' < \deg f'$. Therefore there is at most $O(m\ell \deg f)$ loop iterations from step 2 to 9. □

Corollary 75. *Let $f = (f_1, \ldots, f_\ell) \in M^\ell$ be π-independent vectors. Then we can compute a vector of minimal degree of $\langle f \rangle$ in at most $O(m^3\ell^2 \deg f \operatorname{maxdeg} f)$ arithmetic operations in B and at most $O(rm^2\ell^2 \deg f \operatorname{maxdeg} f)$ multiplications by π in B.*

Proof. This is a direct consequence of Proposition 66 and 74. □

Remark 76. Let $f = (f_1, \ldots, f_\ell) \in M^\ell$ be π-independent vectors. Theorem 66 suggests that one could just first compute $f_\pi = ((f_1)_\pi, \ldots, (f_\ell)_\pi)$ then apply Algorithm 5 and find a shortest vector. But this approach does not provide a shortest vector of minimal filtration as shown by the canonical basis of M^ℓ.

$$\begin{pmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ 1 \\ \vdots \\ 0 \end{pmatrix}, \ldots, \begin{pmatrix} 0 \\ 0 \\ \vdots \\ 1 \end{pmatrix}.$$

The filtration of the shortest vector corresponds to the filtration of the interpolating polynomial of the Sudan and Guruswami-Sudan algorithms. The number of roots depends on the filtration. Let $Q(X, Y) \in B[X, Y]$, then $\pi Q(X, Y)$ has generally more roots than $Q(X, Y)$ (see Chapter 2). As the number of roots can be exponential in the Y-degree of Q it is more desirable to find a short vector with low filtration. We show in the next two examples that the while loop (step 2 to 9) of Algorithm 7 is necessary in order to have a short vector with a low filtration.

Example 77. Let p be a prime and consider the basis

$$f_0 = \left(\begin{pmatrix} pX^4 + pX^2 + X + 1 \\ 0 \end{pmatrix}, \begin{pmatrix} X \\ 1 \end{pmatrix} \right) \in \left(\mathbb{Z}/p^2\mathbb{Z}[X]\right)^2.$$

Applying Algorithm 5, we get

$$f_1 = \left(\begin{pmatrix} pX^2 + X + 1 \\ -pX^3 \end{pmatrix}, \begin{pmatrix} X \\ 1 \end{pmatrix} \right)$$

which is X-reduced. Then we apply Algorithm 6 to obtain

$$f_2 = \left(\begin{pmatrix} pX + p \\ 0 \end{pmatrix}, \begin{pmatrix} X \\ 1 \end{pmatrix} \right).$$

Note that f_2 is not reduced and can be further X-reduced into

$$f_3 = \left(\begin{pmatrix} p \\ -p \end{pmatrix}, \begin{pmatrix} X \\ 1 \end{pmatrix} \right).$$

Therefore it is necessary to perform several X-reductions and π-reductions in order to obtain a shortest vector.

Example 78. Let p be a prime and consider the basis

$$f_0 = \left(\begin{pmatrix} p^2X^5 + X^4 + X \\ 0 \end{pmatrix}, \begin{pmatrix} X^3 \\ pX^2 \end{pmatrix} \right) \in \left(\mathbb{Z}/p^3\mathbb{Z}[X]\right)^2.$$

Applying Algorithm 6, we get

$$f_1 = \left(\begin{pmatrix} pX^4 + pX \\ 0 \end{pmatrix}, \begin{pmatrix} X^3 \\ pX^2 \end{pmatrix} \right)$$

which is p-reduced. Then we apply Algorithm 5 to obtain

$$f_2 = \left(\begin{pmatrix} pX \\ -p^2X^3 \end{pmatrix}, \begin{pmatrix} X^3 \\ pX^2 \end{pmatrix} \right).$$

Note that f_2 is not reduced and can be further p-reduced into

$$f_3 = \left(\begin{pmatrix} p^2X \\ 0 \end{pmatrix}, \begin{pmatrix} X^3 \\ pX^2 \end{pmatrix} \right).$$

Therefore it is necessary to perform several p-reductions and X-reductions in order to obtain a shortest vector.

1.4 Application to list decoding of Reed-Solomon codes

1.4.1 Preliminaries

For any $a, b \in B$ the map

$$\begin{array}{ccc} B[X,Y] & \longrightarrow & B[X,Y] \\ Q(X,Y) & \longmapsto & Q(X+a, Y+b) \end{array}$$

is a ring isomorphism and allows us to define the multiplicity of Q at (a, b).

Definition 79. Let (a, b) be a point of B^2. We say that $Q(X,Y) \in A[X,Y]$ has *multiplicity* m *at* (a, b) if $Q(X + a, Y + b) \in (X,Y)^m$.

Example 80. The polynomial $Q(X,Y) = 5XY \in \mathbb{Z}/5^2\mathbb{Z}$ has multiplicity 2 at $(0,0)$, $(5,0)$, $(0,5)$ and $(5,5)$ as we have

$$5XY = 5 \times (X-5) \times Y = 5 \times X \times (Y-5) = 5 \times (X-5) \times (Y-5).$$

The Sudan or Guruswami-Sudan algorithms can be informally summarized as follow. Given a set of points $\{(x_i, y_i)\}$ of A^2 and some multiplicities $m_i > 0$ one must find a curve of bounded degree that passes through the points (x_i, y_i) with multiplicity at least m_i. To reduce the interpolation step of the Sudan and Guruswami-Sudan list decoding algorithms we need the following lemma. We recall that the degree of a vector $f = (f_1, \ldots, f_m) \in B[X]^m$ is $\max\{\deg f_i : i = 1, \ldots, m\}$.

Lemma 81. *Let $M = (M_{ij}(X)) \in M_{m\times\ell}(B[X])$, Λ be the lattice generated by the columns of M and $f = (f_1, \ldots, f_\ell) \in B[X]^\ell$. Let $k_1, \ldots, k_m$ be positive integers and Γ be the lattice generated by the columns of the matrix*

$$\begin{pmatrix} X^{k_1}M_{11}(X) & \ldots & X^{k_1}M_{1\ell}(X) \\ X^{k_2}M_{21}(X) & \ldots & X^{k_2}M_{2\ell}(X) \\ \vdots & & \vdots \\ X^{k_m}M_{m1}(X) & \ldots & X^{k_m}M_{m\ell}(X) \end{pmatrix}.$$

*Let $f = (f_1, \ldots, f_\ell) \in B[X]^\ell$. Define the $(k_1, \ldots, k_m)$-*column degree of f *to be $\max\{\deg f_i + k_i : i = 1, \ldots, m\}$. Then if $f \in \Gamma$ is a vector of least degree then $f' = (X^{-k_1}f_1, \ldots, X^{-k_m}f_m) \in \Lambda$ and is a vector of least $(k_1, \ldots, k_m)$-column degree.*

1.4.2 Application to the Sudan algorithm

The application to the Sudan algorithm (Algorithm 3) is very simple. We give generating elements of the ideal of all the bivariate polynomials satisfying condition 1 of step 2 of Algorithm 3. The following lemma is the equivalent of [Ale05, Lemma 3.1].

Proposition 82. *Let $x = (x_1, \ldots, x_n) \in B^n$ such that $x_i - x_j \in B^\times$, $y_1, \ldots, y_n \in B$,*

$$Q_0(X) = \prod_{i=1}^{n}(X - x_i) \text{ and } Q_1(X) = \sum_{i=1}^{n} y_i \prod_{j=1}^{n} \frac{X - x_j}{x_i - x_j}.$$

Then the ideal I of all the bivariate polynomials $Q(X, Y) \in B[X, Y]$ such that $Q(x_i, y_i) = 0$ for $i = 1, \ldots, n$ is $Q_0(X)B[X] + (Y - Q_1)B[X, Y]$.

Proof. It is obvious that Q_0 and $(Y - Q_1)$ are in I. Let $Q \in I$. The Euclidean division of Q by $(Y - Q_1)$ gives

$$Q(X, Y) = g(X, Y)(Y - Q_1(X)) + r(X)$$

for $g(X, Y) \in B[X, Y]$ and $r(X) \in B[X]$. Now we have $0 = Q(x_i, y_i) = r(x_i)$ for a fixed i. Thus $(X - x_i)$ divides $r(X)$. As $x_i - x_j \in B^\times$ we have that $Q_0(X)$ divides r(X). □

Taking the notation of Algorithm 3, let $L = \left\lceil \frac{n-\tau-1}{k-1} \right\rceil - 2$. We can seek a polynomial $Q(X, Y)$ satisfying condition 1 of step 2 of Algorithm 3 in the form

$$Q(X, Y) = Q_0(X)h(X) + (Y - Q_1(X)) \sum_{i=1}^{L} g_i(X)Y^i,$$

which corresponds to finding a polynomial of minimal $(0,(k-1),2(k-1),\dots,L(k-1))$-column degree in the lattice generated by the columns of

$$\begin{pmatrix} Q_0(X) & -Q_1(X) & & & \\ & 1 & -Q_1(X) & & \\ & & 1 & -Q_1(X) & \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ & & & & 1 \end{pmatrix}.$$

Thanks to Lemma 81, it corresponds to finding a vector of least degree in the lattice generated by the columns of

$$\begin{pmatrix} Q_0(X) & -Q_1(X) & & & \\ & X^{k-1} & -X^{k-1}Q_1(X) & & \\ & & X^{2(k-1)} & -X^{2(k-1)}Q_1(X) & \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ & & & & X^{L(k-1)} \end{pmatrix}.$$

Algorithm 8 Interpolation step for the Sudan algorithm

Input: A vector $x=(x_1,\dots,x_n)$ such that $x_i-x_j\in B^\times$, a vector $y=(y_1,\dots,y_n)\in B^n$ and an integer L.

Output: A bivariate polynomial $Q(X,Y)$ satisfying conditions 1 and 2 of step 2 of Algorithm 3:

1. $Q(x_i,y_i)=0$ for all $1\le i\le n$.
2. $\deg Q_i\le (n-\tau)-1-i(k-1)$ for all $0\le i\le L$.

1: $Q_0(X)\leftarrow\prod_{i=1}^n(X-x_i)$.
2: $Q_1(X)\leftarrow\sum_{i=1}^n y_i\prod_{j=1}^n\frac{X-x_j}{x_i-x_j}$.
3: $E\leftarrow\emptyset$.
4: $M\leftarrow\begin{pmatrix} Q_0(X) & -Q_1(X) & & & \\ & X^{k-1} & -X^{k-1}Q_1(X) & & \\ & & X^{2(k-1)} & -X^{2(k-1)}Q_1(X) & \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ & & & & X^{L(k-1)} \end{pmatrix}$.
5: Call Algorithm 7 with input $f=$ the columns of M and obtain f'.
6: $r=(r_0,\dots,r_L)\leftarrow$ a vector of f' of least degree.
7: **return** $\sum_{i=0}^L\frac{r_i}{X^{i(k-1)}}Y^i$.

Proposition 83. *Algorithm 8 works correctly as expected with a number of $O(n^7k^2L^3)$ arithmetic operations over B and a number of $O(rn^6k^2L^3)$ multiplications by π.*

- *Or a number of $\tilde{O}(n^7k^2L^3rs\log p)$ bit-operations and $O(rn^6k^2L^3)$ multiplications by p when B is the Galois ring $\mathrm{GR}(p^r,s)$ and*

- *Or a number of* $\tilde{O}(n^7k^2L^3r)$ *arithmetic operations over* $\mathbb{F}_q$ *and* $O(rn^6k^2L^3)$ *multiplications by* t *when* B *is the truncated power series ring* $\mathbb{F}_q[[t]]/(t^r)$.

Proof. This is a direct consequence of Proposition 74, 53 and 54. □

Corollary 84. *For a Reed-Solomon code of parameters* $[n,k]_B$*, one can find the polynomial* $Q(X,Y)$ *satisfying conditions 1 and 2 of step 2 of Algorithm 3 with a number of*

$$O\left(n^7k^2\left(\frac{n}{k}\right)^3\right)$$

arithmetic operations in B *and*

$$O\left(rn^6k^2\left(\frac{n}{k}\right)^3\right)$$

multiplications by π.

- *Or a number of*

$$\tilde{O}\left(n^7k^2\left(\frac{n}{k}\right)^3 rs\log p\right)$$

bit-operations and

$$O\left(rn^6k^2\left(\frac{n}{k}\right)^3\right)$$

multiplications by p *when* B *is the Galois ring* $\mathrm{GR}(p^r,s)$ *and*

- *Or a number of*

$$\tilde{O}\left(n^7k^2\left(\frac{n}{k}\right)^3 r\right)$$

bit-operations and

$$O\left(rn^6k^2\left(\frac{n}{k}\right)^3\right)$$

multiplications by t *when* B *is the truncated power series ring* $\mathbb{F}_q[[t]]/(t^r)$.

Proof. This is a direct consequence of Proposition 83 and step 1 of Algorithm 3. □

1.5 Conclusion

In this chapter we have presented an algorithm to find a vector of minimal degree in a lattice of polynomials over a quotient of a discrete valuation ring. We have designed and presented the so-called "naive algorithm" which corresponds to the "basic algorithm" of Alkhnovich [Ale05]. One has to investigate whether this approach can be done with the Guruswami-Sudan algorithm and if the "improved algorithm" algorithm of [Ale05] can be adapted to our situation, thus obtaining a lower complexity than the ones written in this chapter.

Chapter 2

Polynomial root finding over local rings and application to error correcting codes

This chapter constitutes a submitted work. It has been done in collaboration with Jérémy Berthomieu and Grégoire Lecerf.

Abstract—This article is devoted to algorithms for computing all the roots of a univariate polynomial with coefficients in a complete commutative Noetherian unramified regular local domain, which are given to a fixed common finite precision. We study the cost of our algorithms, discuss their practical performances, and apply our results to the Guruswami and Sudan list decoding algorithm over Galois rings.

2.1 Introduction

Throughout this paper, R denotes a *complete commutative Noetherian unramified regular local domain* of finite dimension r, with maximal ideal $\mathfrak{m}$. Let p denote the characteristic of the residue field $\kappa := R/\mathfrak{m}$ of R, and let $R_i := \mathfrak{m}^i/\mathfrak{m}^{i+1}$, for all $i \geqslant 0$. The fact that R is *unramified* means that either $p = 0$ holds, or that p does not belong to $\mathfrak{m}^2$. By [Coh46, Theorem 15] the following alternative holds:

- If R and κ have the same characteristic whatsoever, then R is isomorphic to the *power series ring* $\kappa[[t_1, \ldots, t_r]]$. In this case, we identify R_i to the subgroup of R of the homogeneous polynomials in $t_1, \ldots, t_r$ over κ of degree i, so that $(R_i)_{i\in\mathbb{N}}$ defines a graduation on R.
- Otherwise, if R and κ have different characteristics, then R is isomorphic to the power series ring $D[[t_1, \ldots, t_{r-1}]]$, where D is a *complete discrete valuation ring* with maximal ideal generated by p. Each element of R can be uniquely written as $\sum_{e\in\mathbb{N}^r} c_e t_1^{e_1} \cdots t_{r-1}^{e_{r-1}} p^{e_r}$, with the c_e in κ. We can still identify R_i to the subset of R of the homogeneous polynomial expressions in $t_1, \ldots, t_{r-1}$ and p of degree i and

with coefficients in κ, but $(R_i)_{i\in\mathbb{N}}$ does not define a graduation on R (for example with R being the ring of the p-adic integers $\mathbb{Z}_p$). In this case, we set $t_r := p$.

In both cases, the function $\nu : R \to \mathbb{N} \cup \{+\infty\}$, which sends 0 to $+\infty$, and any $a \neq 0$ to the largest integer i such that $a \in \mathfrak{m}^i$, is a *valuation*. Any element a of R can be uniquely represented by the converging sum $\sum_{i\geqslant 0}[a]_i$, where $[a]_i \in R_i$ is the *homogeneous component of valuation* i of a. The elements of R_i are called the *homogeneous elements of valuation* i of R.

In this paper we are interested in computing all the roots of a polynomial $F \in R[x]$ given to precision n, which means modulo $\mathfrak{m}^n$. The usual cases are for when $R = \mathbb{Z}_p$ or $R = \mathbb{K}[[t]]$, for any field $\mathbb{K}$. We will adapt classical techniques, analyze their cost, and report on practical performances of our C++ implementation using the MATHEMAGIX libraries [H+02].

2.1.1 Application to list decoding

Univariate polynomial root-finding is a central problem in computer algebra, and a major application resides in decoding certain error-correcting codes as recalled in these paragraphs. Let $a_1, \ldots, a_\lambda$ be λ distinct fixed points in the finite field with q elements, written $\mathbb{F}_q$. Let us recall that a *Reed-Solomon code* of length λ and dimension ρ over $\mathbb{F}_q$ is the set

$$\mathrm{RS}(\lambda, \rho) = \{(f(a_1), \ldots, f(a_\lambda)) : f \in \mathbb{F}_q[x]_{<\rho}\},$$

where $\mathbb{F}_q[x]_{<\rho}$ represents the set of polynomials over $\mathbb{F}_q$ of degree at most $\rho - 1$ (we refer the reader for instance to [Moo05, Chapter 6] for generalities on such codes).

This set $\mathrm{RS}(\lambda, \rho)$ is a vector subspace of $\mathbb{F}_q^\lambda$ of dimension ρ, and there is a one-to-one correspondence between polynomials of $\mathbb{F}_q[x]_{<\rho}$ and elements of $\mathrm{RS}(\lambda, \rho)$. To encode a message, the sender constructs the *unique* polynomial f of $\mathbb{F}_q[x]_{<\rho}$ corresponding to the message, and then transmits the vector $y = (f(a_1), \ldots, f(a_\lambda)) \in \mathbb{F}_q^\lambda$. The received vector may be different from y. If only a few errors occurred during the transmission of y, obtaining the original message can be done using the usual unambiguous decoding algorithms such as Berlekamp-Welch [BW86], Berlekamp-Massey [Ber84], and the extended Euclidean algorithms [TERH88]. But, when more errors occur, a different decoding approach, called *list-decoding*, must be used. A list-decoding algorithm outputs the set Y of all the possible transmitted messages. A postprocess is then responsible for deciding which element of Y is the actual message. Our present motivation lies in the list-decoding algorithms.

In [GS98], *Guruswami and Sudan* designed a polynomial-time list-decoding algorithm. Their method divides into two steps. First it computes a polynomial Q in $\mathbb{F}_q[x][y]$ such that the possible transmitted messages are roots of Q in $\mathbb{F}_q[x]$. In the second step one needs to determine all such roots of Q. Several techniques have been investigated to solve both steps of the problem: see for example [Ale05, AZ08, Köt96, KV03] for the first step and [Joy00, pages 214–228], and [GS00, RR98] for the second step.

The Guruswami and Sudan algorithm has been adapted to other families of codes such as algebraic-geometric codes over fields [GS98], and alternant codes over

fields [ABC10]. Extensions over certain types of finite rings have further been studied for Reed-Solomon and alternant codes in [Arm04, Arm05b], and for algebraic-geometric codes in [Bar06, Wal99a]. In all these cases, the two main steps of the Guruswami and Sudan algorithm are roughly preserved, but to the best of our knowledge, the second step has never been studied into deep details from the complexity point of view. In this paper, we investigate root-finding for polynomials over *Galois rings*, which are often used within these error correcting codes, and that are defined as follows:

Definition 85. Let $\varphi \in \mathbb{Z}/p^n\mathbb{Z}[x]$ be a monic polynomial of degree k that is irreducible modulo p. The ring $(\mathbb{Z}/p^n\mathbb{Z}[x])/(\varphi(x))$ is called the *Galois ring*, written $\mathrm{GR}(p^n, k)$, of *order* nk and *characteristic* p^n.

It is classical that there exists only one Galois ring of order nk and of characteristic p^n up to an isomorphism (see for example [Rag69, p. 207]). On the other hand, notice that such a Galois ring can also be defined as $\mathrm{GR}(p^n, k) = R/(p^n)$, where R is an unramified algebraic extension $\mathbb{Z}_p$ of degree k. Over such a Galois ring $\mathrm{GR}(p^n, k)$ standard techniques cannot be applied to find all the roots of a given polynomial in $\mathrm{GR}(p^n, k)[t][x]$. For instance with $n = 2$ and $F(x) = (x - p)(x - pt)$, one cannot find a value a for t that makes the specialization of F with a unit discriminant in the Galois ring, so that fast classical Newton-Hensel lifting cannot be appealed.

2.1.2 Complexity model

In order to analyze the performances of our algorithms, we denote by $\mathsf{M}(n)$ a cost function for multiplying two univariate polynomials of degree n over an arbitrary commutative ring A with unity, in terms of the number of arithmetic operations in A. Similarly, we denote by $\mathsf{I}(n)$ the time needed to multiply two integers of bit-size at most n in *binary representation*. It is classical [CK91, Für07, SS71] that we can take $\mathsf{M}(n) \in O(n \log n \log\log n)$ and $\mathsf{I}(n) \in O(n \log n 2^{\log^* n})$, where $\log^*$ represents the iterated logarithm of n. Throughout this paper, we assume that $\mathsf{M}(n)/n$ is increasing and that $\mathsf{M}(mn) \leqslant m^2\mathsf{M}(n)$ holds for all positive integers m and n. The same assumptions are also made for I.

When needed, we shall assume that root-finding is computable over the residue field κ. Let us recall here that there exist *effective fields* (that are defined as fields with an effective equality test) for which root-finding is not decidable [FS56, Section 7] (see also another example in [Gat84, Remark 5.10]). Hopefully in most practical cases, roots can be computed efficiently, as we shall recall it later over finite fields.

Finally, let us recall that the *expected cost* spent by a randomized algorithm is defined as the average cost for a given input over all the possible executions. The "soft-Oh" notation $f(n) \in \tilde{O}(g(n))$ means that $f(n) \in g(n) \log^{O(1)}(3 + g(n))$ (we refer the reader to [GG03, Chapter 25, Section 7] for details).

2.1.3 Our contributions

Let $K := \operatorname{Quot} R$ represent the *total field of fractions* of R. Since R is supposed to be complete, so is K, and we still write ν for the extension of the valuation from R to K. The subset of the elements of K of valuation at least i is written O_i. If a is an element of K, and if i is an integer, then we write $a + O_i$ for the set of elements in K whose expansion coincides to those of a to precision i. We say that such a *class* $a+O_i$ is a root of F to precision n if all of its elements annihilate F to precision n. Notice that, for all integers i and j, we have $O_i + O_j = O_{\min(i,j)}$. Thus for any two elements a and b in K we can write $(a + O_i) + (b + O_j) := (a + b) + O_{\min(i,j)}$. By convention, every element a of K can be seen as the class $a + O_\infty$, so that it makes sense to define the sum of an element of K to a class as follows: $a + (b + O_i) := (a + b) + O_i$.

The set of the roots of $F(x) = x^n$ in $\mathbb{Q}_p$ of nonnegative valuation and to precision n is made of all the elements of positive valuation, which amounts to p^{n-1} roots. This simple example shows that the number of roots can be exponential in terms of the size of F. However it can be represented by the single class O_1. In Section 2.2 we show that the roots of nonnegative valuation and to precision n of a polynomial $F \in O_0[x]$ can be represented by at most d such classes, in the sense that the set of roots equals the unions of the elements in these classes. As another example, with $R = \mathbb{Z}_p$, the roots of nonnegative valuation and to precision 4 of $F(x) = x^2(x - 1)$ are either 1 or an element of valuation at least 2 in $\mathbb{Q}_p$, that is in O_2.

Section 2.2 contains a ‘‘naive” algorithm for computing all the roots z of valuation at least a given nonnegative integer w and to a given precision n of a polynomial $F \in O_0[x]$. This algorithm first determines all the possible values for $[z]_w$. Then, from such a value $[z]_w$, it computes the shifted polynomial $F([z]_w + x)$ and it calls itself recursively to obtain the roots of valuation at least $w + 1$. We analyze the complexity of this technique: in particular we show that all subparts but the shifts behave essentially in an optimal way. We also provide the reader with detailed complexity results when R is a univariate power series ring or the p-adic integers ring.

In Section 2.3 we modify the naive algorithm so that it splits the input polynomial between the recursive calls by Hensel lifting. In fact we extend the classical Hensel lifting to the quasi-homogeneous setting, and estimate how it decreases the cost of the previous ‘‘naive” algorithm. We detail complexity bounds when R is a univariate power series ring or the p-adic integers, but also exhibit a probabilistic fast version in higher dimension that avoids expression swell.

Section 2.4 is devoted to applying our root finders in the context of list decoding over Galois rings. We have implemented the algorithms of the present paper when R has Krull dimension 1 in the open source library QUINTIX of the MATHEMAGIX computer algebra system [H+02]. We report on timings and discuss their relative performances.

2.1.4 Related works

Besides the aforementioned works in error correcting codes let us briefly discuss the known materials for computing roots of univariate polynomials over some particular

instances of R as defined from the beginning of the present paper. In both theory and practice, it is classical to compute the factorization, or all the roots in an algebraic closure of a given polynomial $F \in R[x]$ for particular cases. The easiest case is for when the degree of F does not drop modulo $\mathfrak{m}$ and when F is separable modulo $\mathfrak{m}$: Hensel lifting leads efficiently to the unique factorization of F to any requested precision (we refer the reader for instance to [GG03, Chapter 15]). In general, even if F is separable, its residue polynomial modulo $\mathfrak{m}$ may have multiple factors, and one has to make use of the *Newton polygon* technique recursively, assuming that the characteristic is sufficiently large. Over the power series, namely when $R = \mathbb{K}[[t]]$, several authors have contributed to this subject including, for instance: [CC86, CC87, Duv89, Hal01, HM87, PR10, Wal78, Wal99b, Wal00]. Over the p-adic integers the situation becomes more problematic but some of the latter techniques can be extended as in [Hal01]. The case for when R is a power series ring in at least two variables has also been studied in [Iwa05, Kuo89]. In addition, for univariate power series in small characteristic, we refer the reader to [Ked01]. In fact, all these techniques do not solve directly our problem over a general coefficient ring R as considered here, and not even in elementary situations as demonstrated by the following examples:

Example 86. Let $R = \mathbb{Z}_p$, and let $F(x) = (x - p)(x + p)$. In R the polynomial F admits two simple roots p and $-p$, but the set of roots modulo p^2 is the ideal (p). This shows that computing the roots of F in $\mathbb{Z}_p$ does not lead to the ones modulo p^2 directly. In addition the fact that 0 is a root modulo p^2 is contributed by the positive valuation of the values of both factors of F. This suggests that, in general, a kind of exhaustive search might be necessary to recover the modular roots from an irreducible factorization of F in R.

Example 87. Let $R = \mathbb{Z}_p$. The polynomial $F(x) = x^2$ admits 0 as a single double root, but the roots modulo p^4 form the ideal (p^2). Again this shows that there is no obvious relationship between the roots in $\mathbb{Z}_p$ and the ones in $\mathbb{Z}_p/(p^4)$.

These examples illustrate the difficulties for deducing the roots in $R/\mathfrak{m}^n$ from the ones in R to a sufficiently large precision, or from an irreducible factorization over R. The ingredients of the present paper are not substantially new: our main contribution relies in the design of general and well-suited algorithms to the specific root-finding problem.

2.2 Algorithm with linear convergence

Recall that $K := \operatorname{Quot} R$ represents the *total field of fractions* of R. Since R is supposed to be complete, so is K, and we still write ν for the extension of the valuation from R to K. Any element a of K can be uniquely written as the sum $\sum_{i \geqslant \nu(a)} [a]_i$, where $[a]_i$ is 0 or has valuation i and is the quotient of two homogeneous elements in R. For any $i \in \mathbb{Z}$, we write K_i for the set of the elements $a \in K$ such that either a is 0 or a has a single component of valuation i, which means that $a = [a]_i$. The subset of the elements of K of valuation at least i is written O_i.

For any polynomial $F(x) = \sum_{l=0}^{d} F_l x^l \in K[x]$ of degree d, and any $w \in \mathbb{Z}$, we write $[F]_{i,w}$ for the polynomial

$$[F]_{i,w} := \sum_{l=0}^{d} [F_l]_{i-wl} x^l,$$

and call it the *w-homogeneous component of w-valuation i* of F. In addition, the expression $[F]_{j\ldots j+k,w}$ is used to represent the sum $\sum_{l=0}^{k-1} [F]_{j+l,w}$. Remark that if $a \in K$ has valuation at least w then $[F]_{i,w}(a)$ has valuation at least i. Finally the quantity $\nu_w(F)$, called the *w-valuation* of F, stands for the first index $i \in \mathbb{Z}$ such that $[F]_{i,w}$ is nonzero, with the convention that $\nu_w(0) := +\infty$.

Example 88. For $R = \mathbb{Q}[[t]]$, and for $F = x^3 - (1+t)x^2 + t^3$, we have that $\nu_{-1}(F) = -3$, $[F]_{-3,-1} = x^3$, and that $\nu_0(F) = 0$, $[F]_{0,0} = x^3 - x^2$.

2.2.1 Local multiplicities

In this subsection we define the multiplicity of an homogeneous root of a w-homogeneous polynomial.

Lemma 89. (Quasi-homogeneous Euclidean division). *Let $H \in K[x]$ be a non-constant w-homogeneous polynomial of w-valuation i, and let $z \in K_w$. Then there exists a unique w-homogeneous polynomial $Q \in K[x]$ of w-valuation $i-w$, and a unique element $a \in K_i$, such that:*

$$H(x) = [(x - z)Q(x) + a]_{i,w}.$$

Proof. When performing the classical long division of $H(x)$ by $x - z$ the w-homogeneity is preserved in w-valuation i when discarding the carries. □

From the latter lemma, if H is a w-homogeneous polynomial of w-valuation i, then it makes sense to define the *multiplicity* m of any $z \in K_w$ of H, written $\mathrm{mult}(z, H)$, as the largest integer m such that H rewrites into $[(x - z)^m Q(x)]_{i,w}$, where $Q \in K[x]$ is a w-homogeneous polynomial of w-valuation $i - mw$.

Lemma 90. *If $H \in K[x]$ is a nonzero w-homogeneous polynomial of w-valuation i, then the following inequality holds:*

$$\sum_{z \in K_w, H(z) \in O_{i+1}} \mathrm{mult}(z, H) \leqslant \deg H.$$

Proof. Let $z \in K_w$ be of multiplicity m in H, and let $Q \in K[x]$ be as above. If $y \in K_w$ is a distinct root of H to precision $i + 1$, then we have that

$$mv(y - z) + v(Q(y)) \geqslant i + 1.$$

It follows that $v(Q(y)) \geqslant i - mw + 1$, hence that y is a root of Q to precision $i - mw + 1$. By a straightforward induction, we deduce that if $z_1, \ldots, z_s$ are the roots of H in K_w to precision $i + 1$ then H factors into $[(x - z_1)^{m_1} \cdots (x - z_s)^{m_s} G(x)]_{i,w}$, where G is a w-homogeneous polynomial of w-valuation $i - w(m_1 + \cdots + m_s)$, whence the claimed inequality. □

2.2.2 Representation of the set of roots

In this subsection we deal with the representation of sets of truncated roots.

Lemma 91. *Let F be a nonzero polynomial in $K[x]$ of $(w-1)$-valuation i, let $m := \mathrm{mult}(0, [F]_{i,w-1})$, and let $j := \nu_w(F)$. Then we have $i \leqslant j \leqslant i+m$, and $\deg[F]_{j,w} \leqslant j-i \leqslant m$. In addition, $\deg[F]_{j,w} = m$ holds if, and only if, $j = i+m$. In this case the leading coefficients of $[F]_{i,w-1}$ and of $[F]_{j,w}$ coincide.*

Proof. From the assumptions we can express F as $F(x) = x^m Q(x) + H(x)$, where $Q \in K[x]$ is a $(w-1)$-homogeneous polynomial of $(w-1)$-valuation $i - m(w-1)$, such that $Q(0) \neq 0$, and where $H \in K[x]$ is a polynomial of $(w-1)$-valuation at least $i+1$. We see that F has a term ax^m with $\nu(a) = i - m(w-1)$ and $[a]_{i-m(w-1)} = Q(0)$. It follows that the w-valuation j of F is at most $i - m(w-1) + mw = i + m$. On the other hand, since a term of degree $k \geqslant j - i + 1$ in F has $(w-1)$-valuation at least i, it contributes to w-valuation at least $i + k \geqslant j + 1$. Therefore, no monomial of degree at least $j - i + 1$ of F contributes to $[F]_{j,w}$.

If $\deg[F]_{j,w} = m$, then it is clear that $j - i = m$. Conversely, if $j - i = m$ then $[F]_{j,w}$ has the term $[a]_{i-m(w-1)}x^m$, hence has degree m. □

Although the next lemma is elementary, it constitutes the cornerstone of the solver presented in the next subsection.

Lemma 92. *Let F be a nonzero polynomial in $K[x]$ of w-valuation j. Then $a \in K$ is a root of valuation at least w of F to precision n if, and only if, $[F]_{j,w}([a]_w)$ vanishes to precision $j+1$ and $a - [a]_w$ is a root of valuation at least $w+1$ of $F([a]_w + x)$ to precision n.*

Proposition 93. *If F is a polynomial in $O_0[x]$ of w-valuation $j \leqslant n-1$, then its set of roots in K of valuation at least $w \geqslant 0$ and to precision n can be written as the disjoint union of at most $\deg[F]_{j,w}$ classes of the form $a + O_i$.*

Proof. The proof is done by descending induction on w from n. If $w \geqslant n$ then the statement clearly holds since $\deg[F]_{j,w}$ becomes necessarily 0. Let us now assume by induction that the proposition holds for valuation $w+1 \leqslant n$. Let $z \in K_w$ be such that $[F]_{j,w}(z) \in O_{j+1}$, and let $m_z := \mathrm{mult}(z, [F]_{j,w})$. By Lemma 92 the number of classes of roots of F with z as initial term is the number of classes of roots of $F(z+x)$ with valuation at least $w+1$ to precision n. If $j_z := \nu_{w+1}(F(z+x)) \geqslant n$, then there is only one such class. Otherwise the induction hypothesis ensures us that the number of classes is as most $\deg[F(z+x)]_{j_z,w+1}$, which is bounded by m_z by Lemma 91. The conclusion thus follows from Lemma 90. □

2.2.3 Naive local solver

We are to describe an algorithm derived from the proof of Proposition 93. For computational purposes, we need to assume that there exists an algorithm which computes the set of roots in K_w of any w-homogeneous polynomial $H(x)$, together with their respective multiplicities.

Algorithm 9 Naive local solver.

Input: A polynomial $F \in O_0[x]$, $w \in \mathbb{N}$, $i \in \mathbb{N}$, $m \in \mathbb{N}$, $c \in K_{i-(w-1)m}$, and $n \in \mathbb{N}$, such that $i = \nu_{w-1}(F) \leqslant n-1$, $m = \text{mult}(0, [F]_{i,w-1}) \geqslant 1$, and c is the coefficient of degree m in $[F]_{i,w-1}$.

Output: A set of at most m disjoint classes representing the roots of F in K with valuation at least w and to precision n.

1: a. Search for the first nonzero w-homogeneous component H of F taken modulo x^m, of w-valuation j, with $i+1 \leqslant j \leqslant \min(i+m-1, n-1)$.

If such a component does not exist then
If $i+m \leqslant n-1$ then set $j = i+m$ and use c to construct $H = [F]_{j,w}$, otherwise return $\{O_w\}$.
b. If H has degree 0 then return $\{\}$.

2: Compute all the roots $z_1, \ldots, z_s$ in K_w of H to precision $j+1$, together with their respective multiplicities $m_1, \ldots, m_s$.

3: For each e in $1, \ldots, s$ do
a. Compute $F_e := F(z_e + x)$.

b. If $m_e = m$ then let $c_e := c$. Otherwise set c_e to the coefficient of degree m_e in $[F_e]_{j,w}$.

c. Call Algorithm 9 recursively with entries F_e, $w+1$, j, m_e, c_e, and n, in order to obtain the set $Z_{w+1,z}$ representing the roots of F_e of valuation at least $w+1$ to precision n.

3: Return $\{z + z' | z \in Z_w, z' \in Z_{w+1,z}\}$.

Proposition 94. *Algorithm 9 works correctly as specified. In addition, the polynomial H in step 2 of Algorithm 9 equals $[F]_{j,w}$.*

Proof. The algorithm exits at step 1.a with $\{O_w\}$ whenever $\nu_w(F) \geqslant n$, which is correct. It exits at step 1.b with the empty set whenever H is a constant, which is also correct since $H = [F]_{j,w}$ by Lemma 91.

Then the proof is done by descending induction on w. If $w \geqslant n$ then the algorithm necessarily exits at step 1. Let us now assume that the proposition holds for $w+1 \leqslant n$. By Lemma 91 again we have that $H = [F]_{j,w}$. In step 3.b, if $m_e = m$, then Lemma 91 guarantees that c is actually the coefficient of degree m in $[F]_{j,w}$, and thus of $[F_e]_{j,w}$. Therefore the correctness follows from Lemma 92. □

Example 95. Take $R = \mathbb{Q}[[t]]$. The trace of Algorithm 9 with input $F(x) = x^3 - (1+t)x^2 + t^3$, $w = 0$, $i = -3$, $m = 3$, $c = 1$, and $n = 4$ is the following:

1. $j = 0$ and $H(x) = x^3 - x^2$.
2. $z_1 = 0$, $m_1 = 2$, $z_2 = 1$, $m_2 = 1$.
3. Algorithm 9 is called recursively with input $F(0+x) = x^3 - (1+t)x^2 + t^3$, $w = 1$, $i = 0$, $m = 2$, $c = -1$, and $n = 4$, and runs as follows:
   1. $j = 2$ and $H(x) = -x^2$.
   2. $z_1 = 0$, $m_1 = 2$.
   3. Algorithm 9 is called recursively with input $F(0+x) = x^3 - (1+t)x^2 + t^3$, $w = 2$, $i = 2$, $m = 2$, $c = -1$, and $n = 4$, and runs as follows:
      1. $j = 3$, $H(x) = t^3$, and the algorithm returns $\{\}$.
   4. The algorithm returns $\{\}$.

   Algorithm 9 is then called recursively with input $F(1+x) = x^3 + (2-t)x^2 + (1-2t)x - t + t^3$, $w = 1$, $i = 0$, $m = 1$, $c = 1$, and $n = 4$, and runs as follows:

   1. $j = 1$ and $H(x) = x - t$.
   2. $z_1 = t$, $m_1 = 1$.
   3. Algorithm 9 is called recursively with input $F(1+t+x) = x^3 + 2(2+t)x^2 + (1+2t+t^2)x + t^3$, $w = 2$, $i = 1$, $m = 1$, $c = 1$, and $n = 4$, and runs as follows:
      1. $j = 2$ and $H(x) = x$.
      2. $z_1 = 0$, $m_1 = 1$.
      3. Algorithm 9 is called recursively with input $F(1+t+x) = x^3 + 2(1+t)x^2 + (1+2t+t^2)x + t^3$, $w = 3$, $i = 2$, $m = 1$, $c = 1$, and $n = 4$, and runs as follows:
         1. $j = 3$ and $H(x) = x + t^3$.
         2. $z_1 = -t^3$, $m_1 = 1$.

3. Algorithm 9 is called recursively with input $F(1+t-t^3+x) = x^3 + (2 + 2t - 3t^3)x^2 + (1 + 2t + t^2 - 4t^3 - 4t^4 + 3t^6)x - 2t^4 - t^5 + 2t^6 + 2t^7 - t^9$, $w = 4$, $i = 3$, $c = 1$, and $n = 4$, and runs as follows:
   1) The algorithm returns $\{O_4\}$.
4. The algorithm returns $\{-t^3 + O_4\}$.

4. The algorithm returns $\{t - t^3 + O_4\}$.

4. The algorithm finally returns $\{1 + t - t^3 + O_4\}$.

2.2.4 Cumulative cost of steps 1

In step 1 of Algorithm 9, we are interested in counting the cumulative number of extractions of quasi-homogeneous components, and zero tests performed in each graduated component of K. For this purpose we introduce the following subset $T_{i,w-1,m}$ of $\mathbb{N}^2$:

$$T_{i,w-1,m} := \left\{(k,l) \in \mathbb{N}^2 | k \leqslant m-1 \text{and} l \leqslant n-1 \text{and} (w-1)k + l \geqslant i+1\right\}.$$

For any subset S of $\mathbb{N}^2$, we write $|S|$ for its cardinality, and $[S]_v$ for $S \cap (\mathbb{N} \times \{v\})$. Roughly speaking, the following lemma ensures us that the cumulative cost of steps 1 of Algorithm 9 is essentially optimal, whenever an element $a \in O_0$ to precision n is represented as a vector in $K_0 \times \cdots \times K_{n-1}$:

Lemma 96. *For all $v \in \{0, \ldots, n-1\}$, the cumulative number of extractions of homogeneous components of valuation v and the cumulative number of zero tests in each K_v in all steps 1 of Algorithm 9 is at most $|[T_{i,w-1,m}]_v| \leqslant m$.*

Proof. The proof is done by descending induction on w from n down to 0. If $w \geqslant n$ then step 1.a extracts all the components of valuation l of the constant coefficient of F, for $l \geqslant i+1$. The statement therefore holds in this case since $m \geqslant 1$.

Assume that the lemma holds for $w + 1 \leqslant n$. We introduce the auxiliary subset of $\mathbb{N}^2$:

$$S_0 \;:=\; \left\{(k,l) \in \mathbb{N}^2 | k \leqslant m-1 \text{and} l \leqslant n-1 \text{and} i+1 \leqslant (w-1)k + l \text{and} wk + l \leqslant j\right\}.$$

In step 1 of Algorithm 9 only the components of valuation l of the coefficients of x^k for (k,l) in S_0 need to be examined.

Let $M_e := m_1 + \cdots + m_{e-1}$, with the usual convention that $M_1 := 0$. Then, each recursive call for $F(z_e + x)$ in step 3 amounts to at most $|[S_e]_v|$ component extractions and zero tests in K_v, where

$$S_e := (M_e, 0) + T_{j,w,m_e}, \text{for all} e \in \{1, \ldots, s\}.$$

Notice that $S_e \subseteq T_{i,w-1,m}$ holds for all $e \geqslant 0$ by using Lemma 91. On the other hand the S_e are pairwise disjoint. Therefore we obtain that $\sum_{e=0}^{s} |[S_e]_v| \leqslant |[T_{i,w-1,m}]_v|$, which concludes the proof. □

2.2.5 Cumulative cost of steps 2

The following proposition concerns the sum of the degrees of all the polynomials H occurring during the execution of Algorithm 9.

Lemma 97. *The sum of the degrees of all the polynomials H occurring during the execution of all steps 2 of Algorithm 9 does not exceed $m \max(0, n-w)$.*

Proof. The proof is done by descending induction on w from n down to 0. If $w \geqslant n$ then the statement is true since Algorithm 9 exits at step 1. Let us now assume by induction that the lemma holds for $w+1 \leqslant n$. By Lemma 91, each recursive call in step 3 performs root-finding of polynomials whose degree sum does not exceed $m_e(n-(w+1))$. The conclusion thus follows thanks to Lemma 90 as follows:

$$\begin{aligned} m + \sum_{e=1}^{s} (n-(w+1)) m_e &= (n-w)m - (n-(w+1)) \left(m - \sum_{e=1}^{s} m_e \right) \\ &\leqslant (n-w)m. \end{aligned}$$

□

2.2.6 Cumulative cost of steps 3

Let A be any ring. The *shift* of a polynomial $F \in A[x]$ at a point $a \in A$ is the computation of $F(a+x)$. We write $\mathcal{S}_A(d)$ for a bound on the cost of the shift in degree d for $F \in A[x]$ in terms of the number of arithmetic operations in A. We assume that $\mathcal{S}_A(d)/d$ is increasing and that $\mathcal{S}(md) \leqslant m^2 \mathcal{S}(d)$ holds for all positive integers m and d. For the sake of completeness, we briefly recall a classical complexity bound:

Lemma 98. *Let A be a commutative ring with unity, let $F \in A[x]$ be a polynomial of degree d, and let $a \in A$. Then the computation of the shifted polynomial $F(a+x)$ can be done with $O(\mathsf{M}(d) \log d)$ operations in A.*

Proof. We apply the classical divide-and-conquer paradigm. Without loss of generality we can assume that d is a power of 2. We rewrite $F(x)$ into $F_0(x) + x^{d/2} F_1(x)$, with $F_0, F_1 \in A[x]$ of degree at most $d/2$, so that we have $F(a+x) = F_0(a+x) + (a+x)^{d/2} F_1(a+x)$. First we compute all the successive powers $(a+x)^{2^1}, (a+x)^{2^2}, \ldots, (a+x)^{d/2}$, which amounts to $O(\mathsf{M}(d))$ operations in A. Then, the result classically follows from solving the recurrence $\mathcal{S}_A(d) \in 2\mathcal{S}_A(d/2) + O(\mathsf{M}(d/2))$, and with using the assumptions on M. □

Remark 99. Let us mention that the shifted polynomial can be computed faster in some situations. For instance, if $2, 3, \ldots, d$ are invertible in A, and if their respective inverses are given, then one has $\mathcal{S}_A(d) \in O(\mathsf{M}(d))$ by [BP94, Chapter 1, Section 2]. For situations in positive characteristic where the shift can be done within $O(\mathsf{M}(d))$, we refer the reader to [BS05, Proposition 5].

Lemma 100. *Algorithm 9 performs at most $m \max(0, n-w)$ shifts in $O_0[x]$ to precision n.*

Proof. The proof is done by descending induction on w from n down to 0. If $w \geqslant n$ then no shift is performed, so the lemma is true. Let us assume that the lemma holds for $w+1 \leqslant n$. The combination of Lemmas 90 and 91 tells us that the cumulative number of the shifts spent by Algorithm 9 in all steps 3 is at most

$$s + \sum_{e=1}^{s}(n-(w+1))m_e \leqslant (n-w)m + s - \sum_{e=1}^{s} m_e \leqslant (n-w)m.$$

□

For steps 3.b we proceed as for steps 1. We introduce the following subset $T'_{i,w-1,m}$ of $\mathbb{N}^2$:

$$T'_{i,w-1,m} := \left\{(k,l) \in \mathbb{N}^2 | 1 \leqslant k \leqslant m \text{and} l \leqslant n-1 \text{and} (w-1)k + l \geqslant i+1\right\}.$$

The following lemma ensures us that the cumulative cost of steps 3.b of Algorithm 9 is essentially optimal, whenever an element $a \in O_0$ to precision n is represented as a vector in $K_0 \times \cdots \times K_{n-1}$:

Lemma 101. *For all $v \in \{0,\ldots,n-1\}$, the cumulative number of extractions of homogeneous components of valuation v and the cumulative number of zero tests in each K_v in all steps 3.b of Algorithm 9 are at most $|[T'_{i,w-1,m}]_v| \leqslant m$.*

Proof. The proof is done by descending induction on w from n down to 0. If $w \geqslant n$ then the lemma clearly holds since the algorithm exits in step 1. Assume that the lemma holds for $w+1 \leqslant n$, and let $M_e := m_1 + \cdots + m_e$, for $e \in \{1,\ldots,s\}$. If $e = 1$ and $m_1 = m$ then we set $S'_0 := \{\}$, otherwise we set $S'_0 := \{(M_e, j - wm_e) | e = 1,\ldots,s\}$. In step 3.b, when $e \neq 1$ or $m_1 \neq m$ then we associate the component of valuation $j - wm_e$ of the coefficient of x^{m_e} to the point $(M_e, j - wm_e)$ in S'_0.

Then each recursive call for $F(z_e + x)$ in step 3.c amounts to $|[S'_e]_v|$ component extractions and zero tests in K_v, where $S'_e := (M_{e-1}, 0) + T'_{j,w,m_e}$, for all $e \in \{1,\ldots,s\}$. Finally notice that $S'_e \subseteq T'_{i,w-1,m}$ holds for all $e \geqslant 0$ and that all the S'_e are pairwise disjoint. □

2.2.7 Cumulative cost of steps 4

Lemma 102. *The cumulative number of additions of an element of K_v to an element of $K_{v+1} \times \cdots \times K_{n-1}$ performed in all steps 4 of Algorithm 9 is 0 for $v \leqslant w-1$ and at most m for $v \geqslant w$.*

Proof. We prove the lemma by descending induction on w from n down to 0. If $w \geqslant n$ then the lemma is true since step 4 is not reached. Let us now assume by induction that the lemma holds for $w+1 \leqslant n$. If $j \geqslant n$ or if H is a constant then step 4 is not executed. Otherwise by induction and Lemmas 90 and 91, all the recursive calls to Algorithm 9 in step 3 amounts to at most m additions of an element of K_v to an element of $K_{v+1} \times \cdots \times K_{n-1}$ if $v \geqslant w+1$ and 0 otherwise. Then step 4 performs at most m additions of an element of K_w to an element of $K_{w+1} \times \cdots \times K_{n-1}$, which concludes the proof. □

2.2.8 Total cost of Algorithm 9

We assume that κ has either characteristic zero, or admits an algorithm that, for any $k \in \mathbb{N}$, detects if a given element is a p^kth power or not, and returns its p^kth root if it exists. We call this task an *iterated* pth *root extraction*. Let us recall that the *separable decomposition* of a primitive univariate non-constant polynomial G with coefficients in a unique factorization domain A is the decomposition of G into a product $G(x) = \prod_{i=1}^{s} G_i(x^{q_i})^{\mu_i}$, where

- the $G_i \in A[x]$ are primitive, separable, and have positive degrees,
- the $G_i(x^{q_i})$ are pairwise coprime,
- q_i is a power of p if $p > 0$, otherwise $q_i = 1$,
- μ_i is not divisible by p, and the (q_i, μ_i) are pairwise distinct.

The quantity $\sum_{i=1}^{s} \deg G_i$ is called the *separable degree* of G and is denoted by $\operatorname{sdeg} G$. Let us recall that the separable decomposition always exits and is unique up to permutation of the factors and units in A (see for instance [Lec08, Proposition 4]). It coincides to the squarefree decomposition if A has characteristic 0.

From now on, for algorithmic purposes, any element a of R known to precision n is supposed to be stored in dense representation, as the vector $([a]_0, [a]_1, \ldots, [a]_{n-1})$. Any nonzero homogeneous element c of valuation $\nu(c)$ is stored as a vector $(c_e)_{e \in \mathbb{N}^r}$ such that

$$c = \sum_{e \in \mathbb{N}^r, e_1 + \cdots + e_r = \nu(c)} c_e t_1^{e_1} \cdots t_r^{e_r},$$

with all the c_e in κ. Recall that when R and κ have different characteristic then t_r represents p. For such an element c, we write $c^\flat$ for the expression

$$c^\flat := \sum_{e \in \mathbb{N}^r, e_1 + \cdots + e_r = \nu(c)} c_e t_1^{e_1} \cdots t_{r-1}^{e_{r-1}} \in \kappa[t_1, \ldots, t_{r-1}],$$

obtained by substituting 1 for t_r syntactically. If $H(x) = \sum_{l=0}^{d} H_l x^l$ is a w-homogeneous polynomial then we further set $H^\flat(x) := \sum_{l=0}^{d} H_l^\flat x^l$.

Theorem 2. *For any polynomial F in $R[x]$ of degree at most d given to precision n, one can compute a set of at most d disjoint classes representing its set of roots in R to precision n with:*

- *computing primitive parts and separable decompositions of polynomials in $\kappa[t_1, \ldots, t_{r-1}][x]$ of degrees at most d in x and total degrees at most $n-1$ in $t_1, \ldots, t_{r-1}$, and whose degree sum is at most nd,*
- *computing roots in $\kappa[t_1, \ldots, t_{r-1}]$ of at most nd primitive polynomials of degrees 1 and total degrees at most $n-1$ in $t_1, \ldots, t_{r-1}$,*

- *computing roots in* $\kappa[t_1, \ldots, t_{r-1}]$ *of separable polynomials in* $\kappa[t_1, \ldots, t_{r-1}][x]$ *of degrees at least* 2 *and at most* d*, of total degrees at most* $n-1$ *in* $t_1, \ldots, t_{r-1}$*, and whose degree sum is at most* $2(d-1)$*,*
- *extracting iterated* p*th roots of at most* $O(nd/p)$ *elements in* $\kappa[t_1, \ldots, t_{r-1}]$*,*
- $O(nd)$ *shifts of polynomials in* $R[x]$ *of degree at most* d *and to precision* n*, and*
- *an additional number of* $O(d)$ *extractions of homogeneous components of valuation* v*, and zero tests in each* R_v*, for each* $v \in \{0, \ldots, n-1\}$*.*

Proof. Firstly we claim that running Algorithm 9 with input $F \in R[x]$ and finding the only roots in R_w instead of in K_w in step 2 actually leads to the set of roots in R of valuation at least w and to precision n. We leave the proof of this claim to the reader.

We enter this variant of Algorithm 9 with input F, $w = 0$, $i = \nu_{-1}(F)$, $m = \mathrm{mult}(0, [F]_{i,-1})$, n, and the coefficient of degree m of $[F]_{i,-1}$. Determining the values of i and m takes no more than $O(d)$ extractions of homogeneous components of valuation v, and zero tests of elements in each R_v, for $v \in \{0, \ldots, n-1\}$. The cumulative costs of steps 1, 3.b and 4 of Algorithm 9 also drop into $O(d)$ such operations by Lemmas 96, 101, and 102 respectively.

Concerning step 2, we are looking for the roots $z \in R_w$ to precision $j+1$ of $H(x)$. If $H(z) \in O_{j+1}$ then $H^\flat(z^\flat) = 0$ holds in $\kappa[t_1, \ldots, t_{r-1}][x]$ and $z^\flat$ is a polynomial of degree at most w. Conversely, if

$$y = \sum_{e \in \mathbb{N}^{r-1}} y_e t_1^{e_1} \cdots t_{r-1}^{e_{r-1}} \in \kappa[t_1, \ldots, t_{r-1}]$$

has total degree at most w and is a root of $H^\flat(x)$, then we define

$$y^\natural := \sum_{e \in \mathbb{N}^{r-1}} y_e t_1^{e_1} \cdots t_{r-1}^{e_{r-1}} t_r^{w-e_1-\cdots-e_{r-1}} \in R_w,$$

so that $H(y^\natural)$ belongs to O_{j+1}. Therefore, step 2 can be decomposed into the following tasks:

i. Compute the primitive part G of $H^\flat$ and the separable decomposition $G(x) = \prod_{i=1}^{s} G_i(x^{q_i})^{\mu_i}$ seen as in $\kappa[t_1, \ldots, t_{r-1}][x]$,

ii. Compute all the roots in $\kappa[t_1, \ldots, t_{r-1}]$ of all the latter $G_i(x)$,

iii. Extract the necessary q_ith roots of the roots of $G_i(x)$ in order to deduce the ones of $G_i(x^{q_i})$,

iv. Homogenize all the roots y found in iii with t_r, in valuation w, into $y^\natural$ as previously described.

The cumulative cost of tasks i and iii follows from Lemma 97. The cumulative cost of root-finding in ii of polynomials of degree at least 2 follows from Lemma 103 below. Finally the cumulative cost of the shifts in steps 3.a is given in Lemma 100. □

If $G_1, \ldots, G_r$ are polynomials, then we call the quantity $\sum_{e=1}^{r}(\operatorname{sdeg} G_e - 1)$ the *sum of the separable degrees minus* 1 of $G_1, \ldots, G_r$.

Lemma 103. *The sum of the separable degrees minus* 1 *of all the polynomials* $G(x)$ *of steps* i *in the proof of Theorem 2 is at most* $m - 1$.

Proof. The proof is done by descending induction on w. If $w \geqslant n$ then the lemma is true since $m \geqslant 1$ and the algorithm exits in step 1. Let us now assume that the lemma holds for $w + 1 \leqslant n$. If the algorithm exits in step 1 then the lemma is correct. Otherwise, we let m_0 represent the separable degree of $G(x)$. Each recursive call to Algorithm 9 in step 3 performs root-finding of polynomials whose sum of the separable degrees minus 1 does not exceed $m_e - 1$. The total sum of the separable degrees minus 1 is at most

$$\begin{aligned} m_0 - 1 + \sum_{e=1}^{s}(m_e - 1) &\leqslant m_0 - 1 + \sum_{y \in \overline{\kappa(t_1,\ldots,t_{r-1})}, G(y)=0} (\operatorname{mult}(y, G) - 1) \\ &= m_0 - 1 + \deg G - m_0 \\ &\leqslant \deg G - 1. \end{aligned}$$

Finally Lemma 91 provides us with $\deg G - 1 \leqslant m - 1$. □

Corollary 104. *Let* $\mathbb{K}$ *be a field, and let* R *be the power series ring* $\mathbb{K}[[t]]$. *Then, for any polynomial* F *in* $R[x]$ *of degree at most* d *and given to precision* n, *one can compute a set of at most* d *disjoint classes representing its set of roots in* R *to precision* n *with:*

- *computing roots in* $\mathbb{K}$ *of separable polynomials in* $\mathbb{K}[[x]]$ *of degrees at least* 2, *and whose degree sum is at most* $2(d-1)$,
- *extracting iterated* p*th roots of at most* $O(nd/p)$ *elements in* $\mathbb{K}$, *and*
- *an additional number of* $O(nd\mathsf{M}(n)\mathsf{M}(d)\log d)$ *arithmetic operations in* $\mathbb{K}$.

Proof. This is a corollary of Theorem 2. In fact, by [Lec08, Proposition 5], the cumulative cost of the separable factorizations amounts to $O(n\mathsf{M}(d)\log d)$ operations in $\mathbb{K}$. Finally, the cumulative cost of the shifts in steps 3.a is in $O(nd\mathsf{M}(n)\mathsf{M}(d)\log d)$ by Lemma 98. □

Corollary 105. *Let* $\mathbb{K}$ *be a field of characteristic* 0 *and let* R *be the power series ring* $\mathbb{K}[[t]]$. *Then, for any polynomial* F *in* $R[x]$ *of degree at most* d *given to precision* n, *one can compute a set of at most* d *disjoint classes representing its set of roots in* R *to precision* n *with:*

- *computing roots in* $\mathbb{K}$ *of separable polynomials in* $\mathbb{K}[[x]]$ *of degrees at least* 2, *and whose degree sum is at most* $2(d-1)$, *and*
- *an additional number of* $O(nd\mathsf{M}(n)\mathsf{M}(d))$ *arithmetic operations in* $\mathbb{K}$.

Proof. This follows from the previous corollary, by means of Remark 99 that removes a factor of $\log d$ in the cost of the shifts. □

Corollary 106. *Let R be the power series ring $\mathbb{F}_q[[t]]$ over the finite field with $q = p^k$ elements. Then, for any polynomial F in $R[x]$ of degree at most d given to precision n, one can compute a set of at most d disjoint classes representing its set of roots in R to precision n with a randomized algorithm that performs an expected number of*

$$O\left((nd\mathsf{M}(n) + \log q)\mathsf{M}(d)\log d + \frac{nd}{p}\log(q/p)\right)$$

operations in $\mathbb{F}_q$.

Proof. By [GG03, Corollary 14.16] and Corollary 104, the cumulative cost for root-finding amounts to $O(\mathsf{M}(d)\log d\log(dq))$ operations in $\mathbb{F}_q$. □

Let us now focus on the case when R is an unramified algebraic extension of degree $k \geqslant 1$ of the ring $\mathbb{Z}_p$ of the p-adic integers. The ring $R/\mathfrak{m}^n$ is in fact the Galois ring, previously written $\mathrm{GR}(p^n, k)$, in Definition 85. We consider that we are given a monic irreducible polynomial φ in $\mathbb{Z}_p[z]$ of degree k. Let α denote the image of z in R viewed as $(\mathbb{Z}/p^n\mathbb{Z}[z])/(\varphi(z))$. Then, any $a \in R$ can be uniquely written as $\sum_{i=0}^{k-1} a_i\alpha^i$ with $a_i \in \mathbb{Z}/p^n\mathbb{Z}$. We further assume that each a_i is represented by its p-adic expansion $\sum_{j=0}^{n-1} a_{i,j}p^j$, which is stored as the vector $(a_{i,0}, \ldots, a_{i,n-1})$ in $(\mathbb{Z}/p\mathbb{Z})^n$, and where each $a_{i,j}$ is in binary representation. It is classical that the bit-cost for multiplying two elements in $R/\mathfrak{m}^n$ falls in $\tilde{O}(nk\log p)$ [GG03, Chapter 9].

Corollary 107. *Let R be an unramified extension of $\mathbb{Z}_p$ of degree k. Then, for any given polynomial F in $R[x]$ of degree at most d given to precision n, one can compute a set of at most d disjoint classes representing its set of roots in R to precision n with a randomized algorithm that performs an expected number of $\tilde{O}((n^2d + \max(1, n/p)k\log p)dk\log p)$ bit-operations.*

Proof. This is again a corollary of Theorem 2. In fact, by [Lec08, Proposition 5], the cumulative cost of the primitive parts and separable factorizations amounts to $\tilde{O}(nd)$ operations in $\mathbb{F}_q$, where $q := p^k$, which boils down to $\tilde{O}(ndk\log p)$ bit-operations. By [GG03, Corollary 14.16], the cumulative cost for root-finding amounts to $O(\mathsf{M}(d)\log d\log(dq))$ operations in $\mathbb{F}_q$, whence $\tilde{O}(d(k\log p)^2)$ bit-operations. The iterated root extractions take $O\left(\frac{nd}{p}\log(q/p)\right)$ operations in $\mathbb{F}_q$. Finally, the cumulative cost of the shifts in steps 3.a is in $\tilde{O}((nd)^2k\log p)$ by Lemma 98. □

Remark 108. One could decide to store each a_i directly in binary representation modulo p^n: this does not change the latter asymptotic complexity estimate because the change of basis can be computed in softly linear time. In practice this does lightly increase the cost for extracting homogeneous components, but we have shown that these extractions are negligible compared to other operations. Let us mention here that recent practical algorithms on p-adic integers can be found in [BHL10].

2.3 Faster algorithm with splitting

In most situations, the bottleneck of Algorithm 9 resides in the shifts applied on polynomials whose degrees never drop throughout the recursive calls. In this section, we enhance the solver of the previous section by adapting Hensel lifting in order to break the current polynomials into smaller pieces throughout each recursive call.

2.3.1 Quasi-homogeneous Hensel lifting

For any real number $a \in \mathbb{R}$, we write $\lceil a \rceil$ for the smallest integer greater or equal to a. The quasi-homogeneous Hensel lifting algorithm for $F \in K[x]$ summarizes as follows:

Algorithm 10 Quasi-homogeneous Hensel step.

Input: Polynomials F, H_1, H_2, and U in $K[x]$, and integers $w \geqslant 0$, $j \geqslant 0$, and $l \geqslant 1$, such that:

- H_1 is monic of degree d_1, and has w-valuation $j_1 = wd_1$,
- H_2 has degree at most $d_2 := \deg F - d_1$, and w-valuation $j_2 := j - j_1$,
- $[F]_{0\ldots j+l,w} = [H_1 H_2]_{0\ldots j+l,w}$,
- the resultant $\mathrm{Res}(H_1, H_2)$ has valuation $d_1 j_2 = d_1 d_2 w$,
- U has degree at most $d_1 - 1$, w-valuation $-j_2$, and $UH_2 = 1$ holds modulo H_1 and to w-precision $\lceil l/2 \rceil$.

Output: H_1^*, H_2^*, and U^* in $K[x]$ such that:

- H_1^* is monic of degree d_1 and $[H_1^*]_{0\ldots j_1+l,w} = [H_1]_{0\ldots j_1+l,w}$,
- $[F]_{0\ldots j+2l,w} = [H_1^* H_2^*]_{0\ldots j+2l,w}$,
- $U^* H_2^* = 1$ holds modulo H_1^* and to w-precision l.

1: Compute $U^* := (2 - H_2 U)U$ modulo H_1 and to w-precision $-j_2 + l$.
2: Compute $\Delta_F := F - H_1 H_2$ to w-precision $j + 2l$.
3: Compute $\Delta_1 := U^* \Delta_F$ modulo H_1 and w-precision $j_1 + 2l$.
4: Set H_1^* to $H_1 + \Delta_1$, and deduce $H_2^* := F/H_1^*$ to w-precision $j_2 + 2l$.

Algorithm 10 extends the classical Hensel lifting, which specifically concerns the case $w = j_1 = j_2 = 0$ (we refer the reader for instance to [GG03, Chapter 15, Section 4]).

Proposition 109. *Algorithm 10 works correctly as specified. The polynomial H_1^* (resp. H_2^*, U^*) is uniquely determined to w-precision $j_1 + 2l$ (resp. $j_2 + 2l$, l) with the conditions required in the output.*

Proof. It is straightforward to check that $U^*H_2 = 2UH_2 - (UH_2)^2 = 1 - (1 - UH_2)^2 = 1$ holds modulo H_1 and to w-precision l. Let Δ_1 denote an unknown polynomial of w-valuation at least $j_1 + l$, and let Δ_2 denote another unknown polynomial of w-valuation at least $j_2 + l$. By expanding the right-hand side of the equation $F = (H_1 + \Delta_1)(H_2 + \Delta_2)$, we obtain that

$$F - H_1H_2 = H_2\Delta_1 + H_1\Delta_2 + \Delta_1\Delta_2.$$

Truncating the latter expression to w-precision $j + 2l$ leads to

$$[F - H_1H_2]_{j+l\ldots j+2l,w} = [H_2\Delta_1 + H_1\Delta_2]_{j+l\ldots j+2l,w}.$$

By multiplying both hand sides of the latter equation by U^* modulo H_1, we deduce that:

$$[U^*(F - H_1H_2) \bmod H_1]_{j_1+l\ldots j_1+2l,w} = [\Delta_1 \bmod H_1]_{j_1+l\ldots j_1+2l,w}.$$

It follows that Δ_1 exists and is uniquely determined to w-precision $j_1 + 2l$. Therefore H_1^* exists and is uniquely determined as $H_1 + \Delta_1$. Then H_2^* is necessarily determined as F/H_1^* truncated to w-precision $j_2 + 2l$. □

Example 110. Let $R = \mathbb{Z}_p[[t]]$, $F(x) = x^2 - (p^2 + t^2)x + p^2t^2 + t^5$, $w = 2$, $j = 4$, $l = 1$, $H_1(x) = x - p^2$, and $H_2(x) = x - t^2$. We have $d_1 = d_2 = 1$, $j_1 = j_2 = 2$, and $j = 4$. The modular inverse U is $1/(p^2 - t^2)$. We compute $\Delta_F = t^5$, then $\Delta_1 = t^5/(p^2 - t^2)$, and obtain $H_1^*(x) = x - p^2 + t^5/(p^2 - t^2)$, and deduce $H_2^*(x) = x - t^2 - t^5/(p^2 - t^2)$.

Before calling several times Algorithm 10 in order to reach any finite w-precision $j + l$ from w-precision j, one must compute the modular inverse of H_2 modulo H_1, and proceed as summarized in the next algorithm:

Proposition 111. *Algorithm 11 works properly as specified.*

Proof. Since $\mathrm{Res}(H_1, H_2)$ has valuation d_1j_2, the valuation of the inverse of H_2 modulo H_1 as computed in step 1 is exactly $-j_2$. The rest of the proof follows from Proposition 109. □

Corollary 112. *Let F, H_1, and H_2 in $K[x]$ be such that the following conditions hold:*

- *H_1 is monic of degree d_1, and has w-valuation $j_1 = wd_1$,*
- *H_2 has degree at most $d_2 := \deg F - d_1$, and w-valuation $j_2 := j - j_1$,*
- *$[F]_{j,w} = [H_1H_2]_{j,w}$,*
- *the resultant $\mathrm{Res}(H_1, H_2)$ has valuation d_1j_2.*

Then there exist unique polynomials H_1^* and H_2^* in $K[x]$ such that:

- *H_1^* is monic of degree d_1, has w-valuation j_1, and $[H_1^*]_{j_1,w} = [H_1]_{j_1,w}$,*
- *$F = H_1^*H_2^*$.*

Algorithm 11 Quasi-homogeneous Hensel lifting

Input: Polynomials F, H_1, and H_2 in $K[x]$, and integers $w \geqslant 0$, $j \geqslant 0$, and $n \geqslant 1$, such that:

- H_1 is monic of degree d_1, and has w-valuation $j_1 = wd_1$,
- H_2 has degree at most $d_2 := \deg F - d_1$, and w-valuation $j_2 := j - j_1$,
- $[F]_{j,w} = [H_1 H_2]_{j,w}$,
- the resultant $\mathrm{Res}(H_1, H_2)$ has valuation $d_1 j_2 = d_1 d_2 w$.

Output: H_1^*, H_2^* in $K[x]$ such that:

- H_1^* is monic of degree d_1 and $[H_1^*]_{j_1,w} = [H_1]_{j_1,w}$,
- $[F]_{0\ldots j+n,w} = [H_1^* H_2^*]_{0\ldots j+n,w}$.

1: Compute the inverse U of H_2 modulo H_1 in w-valuation $-j_2$.
2: Let $l := 1$, $U^* := U$, $H_1^* := H_1$, and $H_2^* := H_2$.
3: While $l < n$ do

a) Call Algorithm 10 with F, H_1^*, H_2^* and U^*, w, j, and l, and reassign the output into H_1^*, H_2^* and U^* respectively.

b) $l := \min(2l, n)$.

4: Return H_1^* and H_2^*.

In addition, if F belongs to $R[x]$ then $H_2^*(z)H_1^*$ also belongs to $R[x]$, for all $z \in R_w$.

Proof. The existence of H_1^* and H_2^* immediately follows from Proposition 111 since K is complete. As for the last statement, let $z \in R_w$, and let m represents the multiplicity of z in H_1^* ($m = 0$ if z is not a root of H_1^*), let $\tilde{F}(x) := F(x)/(x-z)^m$ and let $\tilde{H}_1^*(x) := H_1^*(x)/(x-z)^m$. Since R is factorial by [Coh46, Theorem 18], Gauss's lemma [Lan02, Chapter IV, Theorem 2.1] ensures us that $\tilde{F}(z)\tilde{H}_1^*/\tilde{H}_1^*(z)$ belongs to $R[x]$. But the latter expression precisely rewrites into $H_2^*(z)\tilde{H}_1^*$, whence $H_2^*(z)H_1^* \in R[x]$. □

Algorithm 10 takes $O(\mathsf{M}(\deg F))$ operations in K. A general cost analysis in terms of operations in κ is difficult since it involves bounding sizes of numerators and denominators of the elements in K used during the intermediate computations. Concerning Algorithm 11, one must in addition describe how the modular inverse of H_2 modulo H_1 is actually obtained. For these reasons, from now on we restrict to considering that the elements of R are represented as in Section 2.2.8. We focus on the important case of dimension 1. Higher dimension is studied in Section 2.3.6.

Lemma 113. *Assume that R has dimension $r = 1$, and let F be a polynomial in $R[x]$ of degree at most d. Then Algorithm 11 can be run so that it performs $O(\mathsf{M}(d)\log d)$ operations in κ, and $O(\mathsf{M}(d))$ operations in $R/\mathfrak{m}^l$, for each value of l in the set $\{1, 2, 4, \ldots, 2^\lambda | 2^\lambda < n\} \cup \{n\}$.*

Proof. The simplest way to implement Algorithm 11 in dimension 1 is to compute $\tilde{F}(x) := F(t_r^w x)/t_r^j$, $\tilde{H}_1(x) := H_1(t_r^w x)/t_r^{j_1}$, $\tilde{H}_2 := H_2(t_r^w x)/t_r^{j_2}$, and $\tilde{U} := U(t_r^w x)/t_r^{-j_2}$, and call Algorithm 11 with input $\tilde{F}$, $\tilde{H}_1$, $\tilde{H}_2$, $w = 0$, $j = 0$, and n. Step 1 can thus be performed by computing an extended g.c.d. between $\tilde{H}_1$ and $\tilde{H}_2$ modulo t_r, which takes $O(\mathsf{M}(d)\log d)$ operations in κ by [GG03, Corollary 11.8]. Then each call to Algorithm 10 can be performed with $O(\mathsf{M}(d))$ operations in R to precision l. Of course at the end we recover H_1^* as $\tilde{H}_1^*(x/t_r^w)t_r^{j_1}$ and H_2^* as $\tilde{H}_2^*(x/t_r^w)t_r^{j_2}$. □

2.3.2 Quasi-homogeneous multifactor Hensel lifting

In this subsection we appeal to the classical divide and conquer paradigm in order to lift any factorization of F into s factors in an efficient way.

Proposition 114. *Algorithm 12 works correctly as specified.*

Proof. The proof follows from induction on s *via* Proposition 111. □

Lemma 115. *Assume that R has dimension $r = 1$, and let F be a polynomial in $R[x]$ of degree d. Then Algorithm 12 can run so that it performs $O(\mathsf{M}(d)\log d \log s)$ operations in κ, and $O(\mathsf{M}(d)\log s)$ operations in $R/\mathfrak{m}^l$, for each value of l in $\{1, 2, 4, \ldots, 2^\lambda | 2^\lambda < n\} \cup \{n\}$.*

Proof. The proof follows from induction on s *via* Lemma 113. □

Algorithm 12 Quasi-homogeneous multifactor Hensel lifting

Input: Polynomials F, $H_1, \ldots, H_{s+1}$ in $K[x]$ and integers $w \geqslant 0$, $j \geqslant 0$, $n \geqslant 1$, such that:

- for all $k \in \{1, \ldots, s\}$, H_k is monic of degree $d_k = \deg H_k$, has w-valuation $j_k = wd_k$,
- H_{s+1} has degree at most $d_{s+1} := \deg F - d_1 - \cdots - d_s$, and has w-valuation $j_{s+1} := j - j_1 - \cdots - j_s$,
- $[F]_{j,w} = [H_1 \cdots H_{s+1}]_{j,w}$,
- For all $k_1 \neq k_2$, the resultant $\mathrm{Res}(H_{k_1}, H_{k_2})$ has valuation $d_{k_1} j_{k_2}$.

Output: $H_1^*, \ldots, H_{s+1}^*$ in $K[x]$ such that:

- for all $k \in \{1, \ldots, s\}$, H_k^* is monic of degree d_k and $[H_k^*]_{j_k,w} = [H_k]_{j_k,w}$,
- $[F]_{0\ldots j+n,w} = [H_1^* \cdots H_{s+1}^*]_{0\ldots j+n,w}$.

1: If $s = 0$ then return $H_1^* := F$.
2: Let $h := \lfloor (s+1)/2 \rfloor$.
3: Compute $G_1 := H_1 \cdots H_h$, and $G_2 := H_{h+1} \cdots H_{s+1}$, $g_1 := j_1 + \cdots + j_h$, and $g_2 := j_{h+1} + \cdots + j_{s+1}$.
4: Call Algorithm 11 with input F, G_1, G_2, w, g_1 and n and let G_1^* and G_2^* denote the output.
5: Call Algorithm 12 recursively with G_1^*, $H_1, \ldots, H_h$, w, g_1, n, and let $H_1^*, \ldots, H_h^*$ be the output.
6: Call Algorithm 12 recursively with G_2^*, $H_{h+1}, \ldots, H_{s+1}$, w, g_2, n, and let $H_{h+1}^*, \ldots, H_{s+1}^*$ be the output.

Algorithm 13 Local solver with splitting.

Input: A polynomial $F \in O_0[x]$, $w \in \mathbb{N}$, $i \in \mathbb{N}$, $m \in \mathbb{N}$, $c \in K_{i-(w-1)m}$ and $n \in \mathbb{N}$, such that $i = \nu_{w-1}(F) \leqslant n-1$, $m = \mathrm{mult}(0, [F]_{i,w-1}) \geqslant 1$, and c is the coefficient of degree m in $[F]_{i,w-1}$.

Output: A set of at most m disjoint classes representing the roots of F in K with valuation at least w and to precision n.

1: a. Search for the first nonzero w-homogeneous component H of F taken modulo x^m, of w-valuation j, with $i+1 \leqslant j \leqslant \min(i+m-1, n-1)$.

If such a component does not exist then

If $i+m \leqslant n-1$ then set $j = i+m$ and use c to construct $H = [F]_{j,w}$, otherwise return $\{O_w\}$.

b. If H has degree 0 then return $\{\}$.

2: Compute all the roots $z_1, \ldots, z_s$ in K_w of H to precision $j+1$, together with their respective multiplicities $m_1, \ldots, m_s$.

3: a. By means of Algorithm 12, compute the factorization of F into $H^*_{s+1} \prod_{e=1}^{s} H^*_e$, where $[H^*_e]_{wm_e,w}(x) = [(x-z_e)^{m_e}]_{wm_e,w}$ for $e \in \{1, \ldots, s\}$.

b. For each e in $1, \ldots, s$ do

i. If $m_e = m$ then let $c_e := c$, and $F_e := F(z_e + x)$.

Otherwise compute $h_e := \prod_{f=1, f\neq e}^{s+1} H^*_f(z_e)$ and let $F_e := h_e H^*_e(z_e + x)$, and $c_e := [h_e]_{j-wm_e}$.

ii. Call Algorithm 13 recursively with entries F_e, $w+1$, j, m_e, c_e and n, in order to obtain the set $Z_{w+1,z}$ representing the roots of F_e of valuation at least $w+1$ to precision n.

4: Return $\{z + z' | z \in Z_w, z' \in Z_{w+1,z}\}$.

2.3.3 Local solver with splitting

In order to decrease the cost of the shifts in Algorithm 9, we modify step 3 as follows:

Proposition 116. *Algorithm 13 works correctly as specified.*

Proof. The algorithm exits at step 1.a with $\{O_w\}$ whenever $\nu_w(F) \geqslant n$, which is correct. It exits at step 1.b with the empty set whenever H is a constant, which is also correct since $H = [F]_{j,w}$ by Lemma 91.

Then the proof is done by descending induction on w. If $w \geqslant n$ then the algorithm necessarily exits at step 1. Let us now assume that the proposition holds for $w+1 \leqslant n$. By Lemma 91 again we have that $H = [F]_{j,w}$. In step 3.b, if $m_e = m$, then Lemma 91 guarantees that c is actually the coefficient of degree m in $[F]_{j,w}$, and thus of $[F_e]_{j,w}$.

Assume that $m_e \neq m$. By construction, $\nu(h_e) = \sum_{f \neq e} \nu(H_f^*(z_e + b)) = j - wm_e$, for all $b \in O_{w+1}$. Therefore an element $b \in O_{w+1}$ is a root of $F(z_e + x)$ to precision n, if, and only if, b is a root of F_e to precision n. The correctness thus follows from Lemma 92. □

Algorithm 13 behaves in the same way as Algorithm 9 regarding to the nature of the recursive calls, to the intermediate values taken by w, i, m, c, and to the successive outputs, as exemplified by running it on the input considered in Example 95:

Example 117. With $R = \mathbb{Q}[[t]]$, here is the trace of Algorithm 13 with input $F(x) = x^3 - (1+t)x^2 + t^3$, $w = 0$, $i = -3$, $m = 3$, $c = 1$, and $n = 4$:

1. $j = 0$ and $H(x) = x^3 - x^2$.

2. $z_1 = 0$, $m_1 = 2$, $z_2 = 1$, $m_2 = 1$.

3. a. Hensel lifting is called with input $F(x)$, $H_1(x) := x^2$, $H_2(x) := x - 1$, $H_3(x) := 1$, $w = 0$, $j = 0$ and $n = 4$. In return we obtain $H_1^*(x) = x^2 - t^3x - t^3$ and $H_2^*(x) = x - 1 - t + t^3$.

   b. Algorithm 9 is called recursively with input $F_1(x) = (-1 - t + t^3)H_1^* = (-1 - t + t^3)x^2 + t^3x + t^3$, $w = 1$, $i = 0$, $m = 2$, $c = -1$, and $n = 4$, and runs as follows:

   1. $j = 2$ and $H(x) = -x^2$.
   2. $z_1 = 0$, $m_1 = 2$.
   3. Algorithm 9 is called recursively with input $F_1(0 + x)$, $w = 2$, $i = 2$, $m = 2$, $c = -1$, and $n = 4$, and runs as follows:
      1. $j = 3$, $H(x) = t^3$, and the algorithm returns $\{\}$.
   4. The algorithm returns $\{\}$.

   Algorithm 9 is then called recursively with input $F_2 = H_2^*(1 + x) = x - t + t^3$, $w = 1$, $i = 0$, $m = 1$, $c = 1$, and $n = 4$, and runs as follows:

   1. $j = 1$ and $H(x) = x - t$.
   2. $z_1 = t$, $m_1 = 1$.

3. Algorithm 9 is called recursively with input $F_2(t+x) = x + t^3$, $w = 2$, $i = 1$, $m = 1$, $c = 1$, and $n = 4$, and runs as follows:
   1. $j = 2$ and $H(x) = x$.
   2. $z_1 = 0$, $m_1 = 1$.
   3. Algorithm 9 is called recursively with input $F_2(t+x)$, $w = 3$, $i = 2$, $m = 1$, $c = 1$, and $n = 4$, and runs as follows:
      1. $j = 3$ and $H(x) = x + t^3$.
      2. $z_1 = -t^3$, $m_1 = 1$.
      3. Algorithm 9 is called recursively with input $F_2(t - t^3 + x) = x$, $w = 4$, $i = 3$, $c = 1$, and $n = 4$, and runs as follows:
         1) The algorithm returns $\{O_4\}$.
   4. The algorithm returns $\{-t^3 + O_4\}$.
4. The algorithm returns $\{t - t^3 + O_4\}$.

4. The algorithm finally returns $\{1 + t - t^3 + O_4\}$.

2.3.4 Total cost of Algorithm 13

Within the same spirit as for Theorem 2, we summarize the cost of Algorithm 13 as follows:

Theorem 3. *For any polynomial F in $R[x]$ of degree at most d given to precision n, one can compute a set of at most d disjoint classes representing its set of roots in R to precision n with:*

- *computing primitive parts and separable decompositions of polynomials in $\kappa[t_1, \ldots, t_{r-1}][x]$ of degrees at most d in x and total degrees at most $n-1$ in $t_1, \ldots, t_{r-1}$, and whose degree sum is at most nd,*
- *computing roots in $\kappa[t_1, \ldots, t_{r-1}]$ of at most nd primitive polynomials of degrees 1 and total degrees at most $n-1$ in $t_1, \ldots, t_{r-1}$,*
- *computing roots in $\kappa[t_1, \ldots, t_{r-1}]$ of separable polynomials in $\kappa[t_1, \ldots, t_{r-1}][x]$ of degrees at least 2 and at most d, of total degrees at most $n-1$ in $t_1, \ldots, t_{r-1}$, and whose degree sum is at most $2(d-1)$,*
- *extracting iterated pth roots of at most $O(nd/p)$ elements in $\kappa[t_1, \ldots, t_{r-1}]$,*
- *multifactor Hensel lifting of polynomials in $R[x]$ of degrees at most d, whose degree sum is at most nd, and to precision n,*
- *$O(n\mathsf{M}(d)\log^2 d)$ operations in R to precision n,*
- *shifts of polynomials in $R[x]$ of degrees at most d, whose degree sum is at most nd, and to precision n, and*

- *an additional number of $O(d)$ extractions of homogeneous components of valuation v, and zero tests in each R_v, for each $v \in \{0, \ldots, n-1\}$.*

Proof. As in the proof of Theorem 2, we claim that running Algorithm 13 with input $F \in R[x]$ and finding the only roots in R_w instead of in K_w in step 2 actually leads to the set of roots in R of valuation at least w and to precision n. This claim can be easily proved by induction thanks to Corollary 112 that ensures that all the F_e in step 3 actually belong to $R[x]$.

We enter this variant of Algorithm 13 with input F, $w = 0$, $i = \nu_{-1}(F)$, $m = \mathrm{mult}(0, [F]_{i,-1})$, n, and the coefficient of degree m of $[F]_{i,-1}$. Determining the values of i and m takes no more than $O(d)$ extractions of homogeneous components of valuation v, and zero tests of elements in each R_v, for $v \in \{0, \ldots, n-1\}$. The computations performed in steps 1 and 4 of Algorithms 9 and 13 are very similar: the successive quantities w, j and n are the same. Therefore the cumulative costs of steps 1 and 4 drops into $O(d)$ extractions of homogeneous components of valuation v, and zero tests of elements in each R_v, for $v \in \{0, \ldots, n-1\}$.

The polynomials H occurring in step 2 of Algorithm 13 are the same of those of Algorithm 9. The cumulative cost of step 2 is thus the same as in the proof of Theorem 2.

Steps 3.a perform multifactor Hensel lifting of polynomials of degree at most m and whose degree sum does not exceed mn by Lemma 97. The same analysis holds for the total cost of the shifts. Finally, the cost for computing all the h_e in steps 3 follows from Lemma 118 below. □

Lemma 118. *Let A be a commutative ring with unity, let $F_1, \ldots, F_s$ be non-constant polynomials in $A[x]$ whose sum of degrees is at most d, and let $a_1, \ldots, a_s$ be in A. Then the computation $\prod_{f=1, f\neq e}^{s} F_f(a_e)$ for $e \in \{1, \ldots, s\}$ can be done with $O(\mathsf{M}(d) \log^2 d)$ operations in A.*

Proof. In order to perform the computation we appeal to the classical divide-and-conquer paradigm:

1. Let $h := \lfloor s/2 \rfloor$. We recursively compute $\prod_{f=1, f\neq e}^{h} F_f(a_e)$ for $e \in \{1, \ldots, h\}$ and then $\prod_{f=h+1, f\neq e}^{s} F_f(a_e)$ for $e \in \{h+1, \ldots, s\}$.

2. We compute $G_1 := F_1 \cdots F_h$ and $G_2 := F_{h+1} \cdots F_s$ with $O(\mathsf{M}(d) \log s)$ operations in A by [GG03, Lemma 10.4].

3. We compute $G_1(a_{h+1}), \ldots, G_1(a_s)$ and $G_2(a_1), \ldots, G_2(a_h)$ with $O(\mathsf{M}(d) \log d)$ operations in A by [GG03, Theorem 10.6].

4. We compute $\prod_{f=1, f\neq e}^{s} F_f(a_e)$ as $G_2(a_e) \prod_{f=1, f\neq e}^{h} F_f(a_e)$ if $e \leqslant h$, and as $G_1(a_e) \prod_{f=h+1, f\neq e}^{s} F_f(a_e)$ otherwise.

The cost function $\mathcal{E}_A(d)$ of this algorithm thus satisfies $\mathcal{E}_A(d) \in \mathcal{E}_A(\deg G_1) + \mathcal{E}_A(\deg G_2) + O(\mathsf{M}(d) \log d)$. We deduce that $\mathcal{E}_A(d) \in O(\mathsf{M}(d) \log^2 d)$. □

As for Algorithm 9, we focus on the case of dimension 1. Remark that in dimension 1 the computation of the h_e in step 3 of Algorithm 13 can be discarded. In fact it suffices to take $h_e := t_r^{j-wm_e}$. The purpose of the h_e is only to ensure that the F_e actually belong to $R[x]$ whenever $r \geqslant 2$.

Corollary 119. *Let $\mathbb{K}$ be a field, and let R be the power series ring $\mathbb{K}[[t]]$. Then, for any polynomial F in $R[x]$ of degree at most d and given to precision n, one can compute a set of at most d disjoint classes representing its set of roots in R to precision n with:*

- *computing roots in $\mathbb{K}$ of separable polynomials in $\mathbb{K}[[x]]$ of degrees at least 2, and whose degree sum is at most $2(d-1)$,*
- *extracting iterated pth roots of at most $O(nd/p)$ elements in $\mathbb{K}$, and*
- *an additional number of $O(n\mathsf{M}(n)\mathsf{M}(d)\log d)$ arithmetic operations in $\mathbb{K}$.*

Proof. This is a corollary of Theorem 3. By [Lec08, Proposition 5], the cumulative cost of the separable factorizations amounts to $O(n\mathsf{M}(d)\log d)$ operations in $\mathbb{K}$. The cumulative cost of the shifts in steps 3 is in $O(n\mathsf{M}(n)\mathsf{M}(d)\log d)$ by Lemma 98. Finally, the cumulative cost of the Hensel liftings in steps 3 is also in $O(n\mathsf{M}(n)\mathsf{M}(d)\log d)$ by Lemma 113. □

Corollary 120. *Let $\mathbb{K}$ be a field of characteristic 0 and let R be the power series ring $\mathbb{K}[[t]]$. Then, for any polynomial F in $R[x]$ of degree at most d given to precision n, one can compute a set of at most d disjoint classes representing its set of roots in R to precision n with:*

- *computing roots in $\mathbb{K}$ of separable polynomials whose degree sum is at most $2(d-1)$, and*
- *an additional number of $O(n\mathsf{M}(n)\mathsf{M}(d)\log d)$ arithmetic operations in $\mathbb{K}$.*

Proof. This follows directly from the previous corollary. □

Corollary 121. *Let R be the power series ring $\mathbb{F}_q[[t]]$ over the finite field with $q = p^k$ elements. Then, for any polynomial F in $R[x]$ of degree at most d given to precision n, one can compute a set of at most d disjoint classes representing its set of roots in R to precision n with a randomized algorithm that performs an expected number of*

$$O\left((n\mathsf{M}(n) + \log(dq))\mathsf{M}(d)\log d + n\frac{d}{p}\log(q/p)\right)$$

operations in $\mathbb{F}_q$.

Proof. By [GG03, Corollary 14.16] and Corollary 119, the cumulative cost for root-finding amounts to $O(\mathsf{M}(d)\log d\log(dq))$ operations in $\mathbb{F}_q$. □

Corollary 122. *Let R be an unramified extension of $\mathbb{Z}_p$ of degree k. Then, for any given polynomial F in $R[x]$ of degree at most d given to precision n, one can compute a set of at most d disjoint classes representing its set of roots in R to precision n with a randomized algorithm that performs an expected number of $\tilde{O}((n + k \log p)ndk \log p)$ bit-operations.*

Proof. This is again a corollary of Theorem 3. In fact, by [Lec08, Proposition 5], the cumulative cost of the primitive parts and separable factorizations amounts to $\tilde{O}(nd)$ operations in $\mathbb{F}_q$, where $q := p^k$, which boils down to $\tilde{O}(ndk \log p)$ bit-operations. By [GG03, Corollary 14.16], the cumulative cost for root-finding amounts to $O(\mathsf{M}(d) \log d \log(dq))$ operations in $\mathbb{F}_q$, whence $\tilde{O}(d(k \log p)^2)$ bit-operations. The iterated root extractions take $O\left(\frac{nd}{p} \log(q/p)\right)$ operations in $\mathbb{F}_q$. Finally, the cumulative cost of the shifts and Hensel liftings in steps 3 is in $\tilde{O}(n^2 dk \log p)$. □

2.3.5 Implementation and timings

In this subsection we compare the performances of Algorithms 9 and 13 for computing all the roots of polynomials F in $\mathbb{Z}/p^n\mathbb{Z}$, where $p := 73$. The family of polynomials F we have taken depends on the parameter d for the degree, n for the precision, and s for the number of roots. In fact F is built as the product of s random monic linear factors times a random polynomial of degree $d - s$.

Our implementation uses the C++ library of MATHEMAGIX [H+02]. It is freely available in the QUINTIX package from the SVN server of MATHEMAGIX at `http://gforge.inria.fr/projects/mmx/`. For the present examples, the root finding for $\mathbb{Z}/p\mathbb{Z}[x]$ uses a naive exhaustive search, which turns out to be very fast whenever p is sufficiently small. Product of polynomials in $\mathbb{Z}/p^n\mathbb{Z}[x]$ is performed *via* the Kronecker substitution [GG03, Chapter 8, Section 4] which reduces to multiplying large integers with GMP [Gra91]. For all the timings we used one core of an Intel(R) Xeon(R) CPU E5520 at 2.27 GHz with 72 Gb of memory, and display timings in milliseconds.

In Tables 2.1 and 2.3 we report on the time spent by Algorithm 9 for various values of d, n and s. Tables 2.2 and 2.4 concern the same computations but performed by Algorithm 13. As expected performances of Algorithm 9 behave roughly quadratically in d, while the ones of Algorithm 13 are roughly linear in d, hence much higher. In these computations we could observe that most of the time of Algorithm 9 is spent in the shifts, while most of the time of Algorithm 13 is spent in Hensel lifting.

d	20	40	80	160	320	640	1280
$s := \lfloor d/2 \rfloor$	4	17	78	380	1623	5802	8527
$s := \lfloor \sqrt{d} \rfloor$	2	5	17	65	242	878	3290

Table 2.1: Algorithm 9 with $R = \mathbb{Z}/73^n\mathbb{Z}$, and $n = 10$.

d	20	40	80	160	320	640	1280
$s := \lfloor d/2 \rfloor$	4	8	18	38	82	178	373
$s := \lfloor \sqrt{d} \rfloor$	2	3	6	12	24	55	113

Table 2.2: Algorithm 13 with $R = \mathbb{Z}/73^n\mathbb{Z}$, and $n = 10$.

d	20	40	80	160	320	640	1280
$s := \lfloor d/2 \rfloor$	409	2191	12212	68944	358565	2120061	10754404
$s := \lfloor \sqrt{d} \rfloor$	166	671	2512	10635	42700	175846	657423

Table 2.3: Algorithm 9 with $R = \mathbb{Z}/73^n\mathbb{Z}$, and $n = 100$.

2.3.6 Cost analysis in higher dimension

When R has dimension $r \geqslant 2$, the naive algorithm has the advantage to operate directly in R, while Algorithm 13 needs to perform divisions in K, which has the drawback to cause an expression swell in the lifting stage. In this subsection we propose a probabilistic approach to avoid this expression swell.

If $a = (a_1, \ldots, a_{r-1})$ is a point in κ^{r-1}, then we write τ_a for the homomorphism from R into R that sends t_i to $(a_i + t_i)t_r$ for all $i \in \{1, \ldots, r-1\}$. If $H(x) = \sum_{l=0}^{d} H_l x^l$ is a polynomial in $R[x]$ then we further set $\tau_a(H)(x) := \sum_{l=0}^{d} \tau_a(H_l)x^l$. Remark that the image of an homogeneous element $c = \sum_{e \in \mathbb{N}^r, e_1+\cdots+e_r=\nu(c)} c_e t_1^{e_1} \cdots t_r^{e_r}$ in R by τ_a is

$$\tau_a(c) = \sum_{e \in \mathbb{N}^r, e_1+\cdots+e_r=\nu(c)} c_e (a_1 + t_1)^{e_1} \cdots (a_{r-1} + t_{r-1})^{e_{r-1}} t_r^{\nu(c)}.$$

Therefore c can be recovered from its value $\tau_a(c)$ if the latter is known to precision $\nu(c)+1$ in t_r and modulo $(t_1, \ldots, t_{r-1})^{\nu(c)+1}$. More generally, if c is any element of R, and if we are given $\tau_a(c)$ to precision $l+1$ in t_r and modulo $(t_1, \ldots, t_{r-1})^{l+1}$, then we can recover c modulo $(t_1, \ldots, t_r)^{\nu(c)+1}$.

Following the discussion on R at the beginning of this article (based on [Coh46, Theorem 15]), if R is the power series ring $\kappa[[t_1, \ldots, t_r]]$ then we let

$$S := \mathrm{Quot}(R/(t_r)) \otimes_{\kappa[[t_r]]} R = \kappa((t_1, \ldots, t_r))[[t_r]].$$

Otherwise, if $R = D[[t_1, \ldots, t_{r-1}]]$, where D is a *complete discrete valuation ring* with maximal ideal generated by $p = t_r$ and residue field $\kappa = D/(p)$, then we let

$$S := \mathrm{Quot}(R/(p)) \otimes_D R = D((t_1, \ldots, t_{r-1})).$$

In both cases, S is a complete commutative Noetherian unramified regular local domain of dimension 1 with maximal ideal $\mathfrak{n} = (t_r)$. We can therefore apply our algorithms in S instead of R as follows:

d	20	40	80	160	320	640	1280
$s := \lfloor d/2 \rfloor$	229	474	984	2085	4431	9615	21135
$s := \lfloor \sqrt{d} \rfloor$	95	151	228	390	676	1346	2616

Table 2.4: Algorithm 13 with $R = \mathbb{Z}/73^n\mathbb{Z}$, and $n = 100$.

Lemma 123. *For any input of Algorithm 11, there exists a nonzero polynomial A in $\kappa[x_1, \ldots, x_{r-1}]$ of degree $d_1 j_2 = w d_1 d_2$ such that, for any point $(a_1, \ldots, a_{r-1}) \in \kappa^{r-1}$ satisfying $A(a_1, \ldots, a_{r-1}) \neq 0$, Algorithm 11 can run on $\tau_a(F)$, $\tau_a(H_1)$, $\tau_a(H_2)$ seen as in $S[x]$, w, j, and n, and returns $\tau_a(H_1^*)$, $\tau_a(H_2^*)$.*

Proof. From the assumptions, $\rho := [\mathrm{Res}(H_1, H_2)]_{d_1 j_2}$ is nonzero. On the one hand, from the specialization property of the resultant, $[\tau_a(\rho)]_{d_1 j_2}$ equals $[\mathrm{Res}(\tau_a(H_1), \tau_a(H_2))]_{d_1 j_2}$. On the other hand, if

$$\rho = \sum_{e \in \mathbb{N}^r, e_1 + \cdots + e_r = d_1 j_2} \rho_e t_1^{e_1} \cdots t_r^{e_r},$$

then $[\tau_a(\rho)]_{d_1 j_2} = \sum_{e \in \mathbb{N}^r, e_1 + \cdots + e_r = d_1 j} \rho_e a_1^{e_1} \cdots a_{r-1}^{e_{r-1}} t_r^{d_1 j}$. We thus let

$$A(x_1, \ldots, x_{r-1}) := \sum_{e \in \mathbb{N}^r, e_1 + \cdots + e_r = d_1 j_2} \rho_e x_1^{e_1} \cdots x_{r-1}^{e_{r-1}}.$$

If $A(a_1, \ldots, a_{r-1}) \neq 0$ then $\tau_a(F)$, $\tau_a(H_1)$, $\tau_a(H_2)$, w, j and n satisfy the requirements of Algorithm 11. □

Lemma 124. *For any input of Algorithm 12, there exists a nonzero polynomial A in $\kappa[x_1, \ldots, x_{r-1}]$ of degree at most $w \deg(F)^2/2$ such that, for any point $(a_1, \ldots, a_{r-1}) \in \kappa^{r-1}$ satisfying $A(a_1, \ldots, a_{r-1}) \neq 0$, Algorithm 12 can run on $\tau_a(F)$, $\tau_a(H_1), \ldots, \tau_a(H_{s+1})$, seen as in $S[x]$, w, j, and n, and returns $\tau_a(H_1^*), \ldots, \tau_a(H_{s+1}^*)$.*

Proof. Let $A_{i,j}$ be the polynomial A of Lemma 123 applied to $H_i H_j$, H_i, H_j, for $i < j$. By the multiplicativity of the resultant it suffices to take $A := \prod_{i<j} A_{i,j}$. The degree of A is $w \sum_{i<j} d_i d_j$, according to the notation of Algorithm 12. The latter sum is bounded by $w \deg(F)^2/2$. □

In order to apply Algorithm 13, it suffices to pick up at random a point $(a_1, \ldots, a_{r-1}) \in \kappa^{r-1}$, then perform the Hensel lifting to precision n in t_r and modulo $(t_1, \ldots, t_{r-1})^n$, compute $\tau_a(F_e)$, and finally recover F_e in $R[x]$ since it actually belongs to $R[x]$. In this way, if κ has sufficiently many elements, then Algorithm 13 behaves efficiently in high dimension with a high probability of success.

2.4 Application to error correcting codes

Let E be an unramified extension of $\mathbb{Z}_p$ of degree k so that $E/(p^n)$ is the Galois ring $\mathrm{GR}(p^n, k)$ of Definition 85, and let $q := p^k$.

2.4.1 Algorithm

Let F be a polynomial in $E[t][x]$ of degree at most d in x and degree at most d_t in t. We are interested in computing all the roots of F in $E[t]$ of degree at most a given integer l, and modulo p^n.

Algorithm 14 Root finding for bivariate polynomials.

Output: A polynomial $F \in E[t][x]$ of degree at most d in x and d_t in t, and two nonnegative integers n and l.

Input: A set of at most d disjoint classes representing the roots of F in $E[t]$ of degree at most l modulo p^n.

1: Compute an irreducible polynomial $\varphi(t) \in \mathbb{F}_q[t]$ of degree $e = dl + d_t + 1$.
2: Call Algorithm 9 or 13 with $R := E[t]/(\varphi(t))$, and F seen in $R[x]$ of degree at most d, in order to obtain a set Z of at most d disjoint classes of the roots.
3: Return the elements of Z of degree at most l in t.

Proposition 125. *Algorithm 14 works correctly, and takes:*

- *an expected number of* $\tilde{O}((n^2 d + \max(1, n/p)ek \log p)dek \log p)$ *bit-operations when using the naive solver derived from Algorithm 9, or*
- *an expected number of* $\tilde{O}((n^2 + nek \log p)dek \log p)$ *bit-operations when using the naive solver derived from Algorithm 13.*

Proof. A polynomial $z(t)$ is a root of F of degree at most l modulo p^n if, and only if, it is a root of F seen in $E[t]/(\varphi(t))$ modulo p^n, since $F(z(t))$ has degree at most $dl + d_t$.

Step 1 can be done with an expected number of $\tilde{O}(e^2 \log q)$ operations in $\mathbb{F}_q$ by [GG03, Corollary 14.43]. The cost of step 2 then follows from Corollary 107 (resp. from Corollary 122) when using Algorithm 9 (resp. using Algorithm 13). □

2.4.2 Experiments

We have implemented finite fields in the C++ package of MATHEMAGIX called FINITEFIELDZ. Several representations and algorithms are available, including products *via* lookup tables for small fields, a wrapper of the MPFQ library [GT06] for specific fields, and a generic implementation as quotient ring for larger fields. We have also implemented Galois rings in the aforementioned QUINTIX package, in a way very similar to finite fields. Root finding can be performed either by an exhaustive search or *via* Berlekamp or Cantor-Zassenhaus based algorithms (see for instance [GG03, Chapter 14]).

Algorithm 14 is available in the QUINTIX package. In order to test it, we built input polynomials from real examples by using Sudan's interpolation algorithm for Reed-Solomon codes over Galois rings [Sud97b, Lemma 4]. This interpolation relies merely on linear algebra over Galois rings as described in [Arm04, Arm05b]. In Tables 2.5 and 2.6 we display the performances of Algorithm 14 for various length of the code. Timings are

Length of the code	100	200	250
$\mathbb{Z}/p^{10}\mathbb{Z}$	$\mathbb{Z}/103^{10}\mathbb{Z}$	$\mathbb{Z}/211^{10}\mathbb{Z}$	$\mathbb{Z}/257^{10}\mathbb{Z}$
d	3	2	2
d_t	29	116	83
l	9	49	59
e	57	215	202
Algorithm 9 in step 2 (ms)	785	5492	3509
Algorithm 13 in step 2 (ms)	1068	10298	14978

Table 2.5: Algorithm 14 for Reed-Solomon codes over $\mathbb{Z}/p^{10}\mathbb{Z}$.

Length of the code	100	200	250
$\mathbb{Z}/p^{100}\mathbb{Z}$	$\mathbb{Z}/103^{100}\mathbb{Z}$	$\mathbb{Z}/211^{100}\mathbb{Z}$	$\mathbb{Z}/257^{100}\mathbb{Z}$
d	3	2	2
d_t	29	116	83
l	9	49	59
e	57	215	202
Algorithm 9 in step 2 (ms)	675	11421	6134
Algorithm 13 in step 2 (ms)	2046	9942	10861

Table 2.6: Algorithm 14 for Reed-Solomon codes over $\mathbb{Z}/p^{100}\mathbb{Z}$.

measured in milliseconds in the same conditions as in Section 2.3.5, and we compare the relative performances of Algorithms 9 and 13.

Notice that the timings are somehow similar between precision 10 and 100. This is mainly because the interpolation step returns a polynomial whose coefficients have valuations close to the precision. Moreover the degrees in x being very small compared to the extension degree of the Galois ring used by Algorithm 14 in step 2, both Algorithms 9 and 13 spend a lot of time in the root-finding algorithm over large finite fields.

In the latter examples, we can see that the degree d is rather small in comparison to d_t. Heuristically, this fact could be related to [NH00, Proposition 12 page 9] which states that the probability of having more than one codeword in a Hamming ball, whose radius corresponds to the Sudan algorithm decoding radius, is close to zero. The degree d of F is related to the number c of codewords within the Hamming ball by $c \leqslant d$. And, in practice, we observe that d is close to 1 when $c = 1$ with probability close to 1.

Of course one can construct received words such that the decoding algorithm has to return a given number c of codewords. Hence, by the inequality $c \leqslant d$, one can force the degree d to be at least a given positive integer. Such a word can be built as follows. First denote by $(r)_{i\cdots j}$ the vector $(r_i, r_{i+1}, \ldots, r_j)$ for any vector r with coefficients in $\mathbb{Z}/p^n\mathbb{Z}$. Take d codewords $c_1, \ldots, c_d$ such that $(c_1)_{1\cdots k-d-1} = (c_2)_{1\cdots k-d-1} = \cdots = (c_d)_{1\cdots k-d-1}$ where k is the rank of the code. Then compute $\Delta = \lfloor (\ell - k - d)/d \rfloor$ where ℓ is the length of the code. Finally compute the word

$$\rho = ((c_1)_{1\cdots k-d-1}; (c_1)_{k-d\cdots k-d+\Delta}; (c_2)_{k-d+\Delta+1\cdots k-d+2\Delta}; \cdots; (c_d)_{k-d+(d-1)\Delta\cdots k-d+d\Delta}),$$

and truncate ρ, if necessary, so that its length equals the length ℓ of the code. Table 2.7 reports on timings obtained with this construction.

Length of the code	1400	1800	2000
$\mathbb{Z}/p^{10}\mathbb{Z}$	$\mathbb{Z}/1409^{10}\mathbb{Z}$	$\mathbb{Z}/1811^{10}\mathbb{Z}$	$\mathbb{Z}/2003^{10}\mathbb{Z}$
d	6	8	10
d_t	43	50	45
l	9	9	9
e	98	123	136
Algorithm 9 in step 2 (ms)	19742	58414	145380
Algorithm 13 in step 2 (ms)	23818	70941	140828

Table 2.7: Algorithm 14 for Reed-Solomon codes over $\mathbb{Z}/p^{10}\mathbb{Z}$ with a forced degree d for the interpolation polynomial F.

Notice that Algorithm 13 starts to be interesting when the degree d is at least 10 for codes with a very low rate. In this case the code rate is smaller than 0.5%. Therefore the naive algorithm turns out to be sufficient for practical applications whenever the code rate is close to 1.

Acknowledgments

We would like to thank Daniel Augot for his useful comments on this article.

Part II

A Lifting Framework for List Decoding over some Finite Rings

Context

In this part we exploit the structure of some finite rings (such as the Galois rings) to obtain decoding algorithms over rings from decoding algorithms over finite fields. We first recall the definitions of Galois rings and generalized Reed-Solomon codes over rings.

Proposition. *Let p be a prime and r, s two positive integers. Let $\varphi(X), \phi(X) \in \mathbb{Z}/p^r\mathbb{Z}[X]$ be two degree-s monic polynomials irreducible modulo p. Then there is a ring isomorphism*

$$\frac{\mathbb{Z}/p^r\mathbb{Z}[X]}{(\varphi(X))} = \frac{\mathbb{Z}/p^r\mathbb{Z}[X]}{(\phi(X))}.$$

The proof can be found in [Rag69, Statements I and II, page 207].

Definition (Galois rings)**.** The ring $\frac{\mathbb{Z}/p^r\mathbb{Z}[X]}{(\varphi(X))}$ from the previous proposition is denoted by $\mathrm{GR}(p^r, s)$ and called a *Galois ring.*

Definition. Let $k < n$ be two positive integers, $v = (v_1, \ldots, v_n)$ be such that $v_i \in Z(A)^\times$ for all i and $x = (x_1, \ldots, x_n) \in A^n$ be such that for all $i \neq j$, $x_i \neq x_j$ and $x_i x_j = x_j x_i$. The *left submodule* generated by the

$$(v_1 f(x_1), \ldots, v_n f(x_n)) \in A^n, f \in A[X] \text{ and } \deg f < k$$

is called a *generalized Reed-Solomon code of parameters* $[n, k]_A$ and denoted by $\mathrm{GRS}_A(v, x, n)$ or simply $\mathrm{GRS}(n, k)$ when there is no confusion. The vector x is called the *support* while v is called the *weight* of $\mathrm{GRS}_A(v, x, k)$.

Let A be a ring with identity with the following property:

$(*)$ *there exists a regular element p which is not a unit such that $p \in Z(A)$ and such that every element $a \in A$ can be uniquely written as $\sum_{i=0}^{\infty} a_i p^i$ where, for all $i \in \mathbb{N}$, a_i is in a set of representatives of $A/(p)$.*

For example A can be

- the power series ring over a field κ, $\kappa[[t]]$,
- an unramified extension of degree s of the p-adic ring $\mathbb{Z}_p$, $\mathbb{Z}_{p^s}$.
- the matrix ring over $\kappa[[t]]$, $M_\ell(\kappa[[t]])$,
- the matrix ring over $\mathbb{Z}_{p^s}$, $M_\ell(\mathbb{Z}_{p^s})$.

Let r be a positive integer and $B = A/(p^r)$ and let $\mathcal{C}$ be a generalized Reed-Solomon code over B of parameters $[n, k]_B$. The idea behind the algorithms in this part is to take advantage of the decoding algorithms of generalized Reed-Solomon codes over a finite fields which have been widely studied in [Ber84, BW86, TERH88] for the unique decoding and in [Köt96, Sud97b, RR98, GS98, Ale05, AZ08] for the list decoding. The algorithm is a lifting algorithm that has the advantage to be very simple. We give an example of the trace of the algorithm.

Example 126. Let $B = \mathbb{Z}/5^2\mathbb{Z}$, $x = (1, 2, 3, 4)$ and $\mathcal{C} = \mathrm{RS}_B(x, 2)$. Suppose that $y = (6, 12, 6, 6)$ is the received word. The first step is to consider $y_1 = y \mod 5 = (1, 2, 1, 1)$ seen as a received word. A decoding algorithm for $\mathrm{RS}_{\mathbb{F}_5}(x \mod 5, 2)$ gives the nearest (within distance 1 from y_1) codeword $c_1 = (1, 1, 1, 1)$ and the corresponding error vector $e_1 = (0, 1, 0, 0)$. Then compute

$$\begin{aligned} y \longleftarrow \text{“}(y - y_1 - e_1)/5 &= [(6, 12, 6, 6) - (1, 1, 1, 1) - (0, 1, 0, 0)]/5\text{”} \\ &= (1, 2, 1, 1). \end{aligned}$$

Then, again, we consider $y_2 = (1, 2, 1, 1) \mod 5 = (1, 2, 1, 1)$ and a unique decoding algorithm for $\mathrm{RS}_{\mathbb{F}_5}(x \mod 5, 2)$ gives the codeword $c_2 = (1, 1, 1, 1)$ and the corresponding error vector $(0, 1, 0, 0)$. The reconstruction of the transmitted codeword (within distance 1 of $y = (6, 12, 6, 6)$) and its corresponding error vector of $\mathbb{F}_5^4$ is easy.

$$\begin{aligned} c &= (1, 1, 1, 1) + 5 \cdot (1, 1, 1, 1) = (6, 6, 6, 6) \\ e &= (0, 1, 0, 0) + 5 \cdot (0, 1, 0, 0) = (0, 6, 0, 0) \end{aligned}$$

The lifting algorithm has been first proposed in [GV98] then studied in [Byr01,BZ01]. I then applied it to interleaved linear codes using erasures to improve its decoding radius.

Definition 127. We let A be the power series ring over the finite field $\mathbb{F}_q$ and $B = \mathbb{F}_q[[t]]/(t^r)$. We let $\mathcal{C}$ be a linear code over $\mathbb{F}_q$ with parameters $[n, k, d]_{\mathbb{F}_q}$ and with generator matrix G. Let r messages $m_0, \ldots, m_{r-1} \in \mathbb{F}_q^k$ and their encoding $c_0 = m_0G, \ldots, c_{r-1} = m_{r-1}G$. For $i = 0, \ldots, r-1$ and $j = 1, \ldots, n$ define c_{ij} to be the j-th coordinate of c_i and $s_j = (c_{0,j}, \ldots, c_{r-1,j})$.

$$\begin{array}{|c|c|c|c|} \hline c_{0,1} & c_{0,2} & \cdots & c_{0,n} \\ \hline c_{1,1} & c_{1,2} & \cdots & c_{1,n} \\ \hline \vdots & \vdots & \vdots & \vdots \\ \hline c_{r-1,1} & c_{r-1,2} & \cdots & c_{r-1,n} \\ \hline \end{array} \begin{array}{l} \to \ c_0 \\ \to \ c_1 \\ \\ \to \ c_{r-1} \end{array}$$

$$\begin{array}{cccc} \downarrow & \downarrow & & \downarrow \\ s_1 & s_2 & & s_n \end{array}$$

The vectors transmitted over the channel are not $c_1, \ldots, c_{r-1} \in \mathbb{F}_q^n$ but $s_1, \ldots, s_n \in \mathbb{F}_q^r$. We will make an abuse of notation and call such an encoding scheme a *interleaved code with respect to* $\mathcal{C}$ *and of degree* r. In this context a *burst error* is an error occurring in one whole column s_i for one index $i \in \{1, \ldots, n\}$.

Interleaved linear codes over finite fields are presented in [VVO89, Chapter 7, Section 5] and their decoding is studied in [BKY03, CS03, GGR11]. They have also been considered over Galois rings in [Arm10]. The improvement of the lifting algorithm is made with erasures following an idea of M. A. Armand.

Contributions

No complexity was given for the lifting algorithm. It was given only for quotient of discrete valuation rings which are commutative rings, it has not been studied in the situation of list decoding and it has not been used outside the unique decoding of codes. First, in Chapter 3 we show that this lifting technique works for any generalized Reed-Solomon code over any finite ring satisfying $(*)$ and study its complexity in the situation of unique and list decoding. Then in Chapter 4, I show how to improve the lifting algorithm for decoding linear codes over B and I show how to apply it to interleaved linear codes over a finite fields in which case it permits to correct, with high probability, more than τ burst errors.

Chapter 3

On Generalized Reed-Solomon Codes Over Commutative and Noncommutative Rings

This chapter constitutes a submitted work. It has been done in collaboration with Morgan Barbier and Christophe Chabot.

Abstract—In this paper we study generalized Reed-Solomon codes (GRS codes) over commutative, noncommutative rings, show that the classical Welch-Berlekamp and Guruswami-Sudan decoding algorithms still hold in this context and we investigate their complexities. Under some hypothesis, the study of noncommutative generalized Reed-Solomon codes over finite rings leads to the fact that GRS code over commutative rings have better parameters than their noncommutative counterparts. Also GRS codes over finite fields have better parameters than their commutative rings counterparts. But we also show that given a unique decoding algorithm for a GRS code over a finite field, there exists a unique decoding algorithm for a GRS code over a truncated power series ring with a better asymptotic complexity. Moreover we generalize a lifting decoding scheme to obtain new unique and list decoding algorithms designed to work when the base ring is for example a Galois ring or a truncated power series ring or the ring of square matrices over the latter ring.
Keywords—Algebra, Algorithm design and analysis, Decoding, Error correction, Reed-Solomon codes.

3.1 Introduction

Reed-Solomon codes (denoted by RS codes in the rest of this paper) form an important and well-studied family of codes. They were first proposed in 1960 by Irvin Stoy Reed and Gustave Solomon in their original paper [RS60]. They have the property to be *Maximum Distance Separable* (MDS) codes, thus reaching the Singleton bound. Not only do RS codes have the best possible parameters, they also can be efficiently decoded. See for

example [Gao02] and [Jus76]. They are widely used in practice such as in compact disc players, disk drives, satellite communications, and high-speed modems such as ADSL. See [WB99] for details about applications of RS codes. A breakthrough has been made by Madhu Sudan in 1997 about the list decoding of RS codes in his paper [Sud97b], further improved by Venkatesan Guruswami and Madhu Sudan in [GS98]. They showed that RS codes are list decodable up to the generic Johnson bound in polynomial time. In [NH00], Rasmus Refslund Nielsen and Tom Høholdt showed that the probability of having more than one codeword returned by any list decoding algorithm which can decode up to the generic Johnson bound is very small, making the Guruswami-Sudan algorithm usable in practice.

In the present article we investigate generalized Reed-Solomon codes (denoted by GRS codes in the rest of this paper) over rings with unity. The latter need not be commutative. We show that the main results about GRS codes still hold in this more general situation.

3.1.1 Our contributions

In an attempt to build new good codes over $\mathbb{Z}/4\mathbb{Z}$ in a similar way as in [BCQ12] we noticed that most results about evaluation codes still hold when we replace the ring of matrices by any noncommutative ring. This generalization is natural to do and permits the study of GRS codes over rings with unity in great generality.

Moreover it allows us to design unique and list decoding algorithms, prove their correctness and study their asymptotic complexities in a very general framework. They are not constraint to have as input a GRS code over a finite field or a Galois ring. They remain valid when the alphabet is any noncommutative ring such as matrices over a finite field or a Galois ring.

In this article we reach the conclusion that GRS codes over finite noncommutative rings are no better than GRS codes over finite commutative rings which are themselves no better than their finite fields counterparts as far as *only* the parameters are concerned.

We summarize our results in the following theorems and propositions which will be proved later in the article.

Theorem 128. *Given three positive integers $k < n \leq q$, let A be a non commutative ring of cardinality q and a GRS code over A of parameters $[n, k, n-k+1]_A$. Then there exists a commutative ring B of cardinality q and a GRS code over B of parameters $[n, k, n-k+1]_B$.*

The same theorem holds when q is a prime power and if we replace "noncommutative rings" by "commutative rings" and "commutative rings" by "finite fields". The "soft-Oh" notation $f(n) \in \tilde{O}(n)$ means that $f(n) \in g(n)\log^{O(1)}(3+g(n))$ (we refer the reader to [GG03, Chapter 25, Section 7] for details).

Proposition 129. *Given a Galois ring $A = \mathrm{GR}(p^r, s)$ and a RS code over A with parameters $[n, k, n-k+1]_A$, there exists a unique decoding with an asymptotic complexity of $\tilde{O}(rnks\log p)$ bit-operations; and a list decoding algorithm with an asymptotic*

complexity of $\tilde{O}(n^{r+6}k^5sp^{rs(r-1)})$ *bit-operations which can list decode up to the Johnson bound.*

In this paper we provide detailed asymptotic complexities of our decoding algorithms when the alphabet of the RS code is a Galois ring and the ring $\mathbb{F}_q[[t]]/(t^r)$. We denote by $\mathfrak{D}$ the unique decoding algorithm that can be found in [Ber68, Mas69, SKHN75, Jus76, BW86, Gao02].

Theorem 130. *Given a finite field* A*, a truncated power series ring* B *such that* $|A| = |B|$*, a RS code* $\mathcal{C}_A$ *over* A *of parameters* $[n, k, n-k+1]_A$ *and a unique decoding algorithm* `UDec` *from the list* $\mathfrak{D}$ *for* $\mathcal{C}_A$*. Suppose that there exists a RS code* $\mathcal{C}_B$ *over* B *of parameters* $[n, k, n-k+1]_B$*. Then there exists a RS code* $\mathcal{C}'_B$ *over* B *of parameters* $[n, k, n-k+1]_B$ *such that* $\mathcal{C}_B/p\mathcal{C}_B = \mathcal{C}'_B/p\mathcal{C}'_B$ *and a unique decoding algorithm for* $\mathcal{C}'_B$ *with a better asymptotic complexity than* `UDec` *as soon as the complexity of* `UDec` *is equal or greater than* $\tilde{O}(n)$*.*

Note that the asymptotic complexity of the known unique decoding algorithms is at least $\tilde{O}(n)$. In addition, we show that the gain is more significant when the arithmetic of the underlying rings is not done with asymptotically fast algorithms which is the case for practical applications. In this case we have a similar theorem as Theorem 130 for Galois rings.

3.1.2 Related work

Our approach for building noncommutative GRS codes is different from the one to build "skew codes" [BGU07, BSU08, BU09a, BU09b, CLU09]. Skew polynomial rings over finite fields or Galois rings are used for the construction of codes whose alphabets are finite fields or Galois rings. Here we consider alphabets which are noncommutative rings and not necessarily finite fields. GRS codes over a commutative finite ring have been studied by Marc André Armand in [Arm04, Arm05b]. To our knowledge this paper is the first to study GRS codes over noncommutative rings.

Unique decoding algorithms for RS codes over finite fields have been studied for example in [Ber84, BW86, TERH88]. List decoding algorithms for RS codes over finite fields have been investigated for example in [Ale05, AZ08, GS98, Köt96, KV03, RR98, Sud97b]. A unique decoding algorithm for RS codes over Galois rings has been proposed by [Arm05c] while list decoding algorithms have been investigated in [Arm04, Arm05b, Arm05a, AdT05].

A lifting decoding scheme has been first proposed in [GV98] then in [Byr01, BZ01]. In this paper we generalize the lifting decoding scheme to obtain unique and list decoding algorithms for GRS code over noncommutative rings.

3.2 Prerequisites

In this article we let A be a (*not necessarily* commutative) ring with unity, denoted by 1 or 1_A, such that for all $a, b \in A$

- $1.a = a.1 = a$,
- $a.b = 1 \Longrightarrow b.a = 1$.

If $ab = 1$, we say that a is *invertible* or that a is a *unit* whose inverse is b. In this paper, we will only consider rings verifying the above conditions and by "ring" we mean a *not necessarily commutative* ring unless stated otherwise. We denote by $A^\times$ the *not necessarily commutative* group of units of A. An element $a \in A$ is *right regular* if $ab = 0$ implies $b = 0$ for all $b \in A$. Similarly a is *left regular* if $ba = 0$ implies $b = 0$ for all $b \in A$. If a is both left and right regular we say that a is regular. Note that when A is finite, $A^\times$ coincides with the group of regular elements of A. Suppose now that we have $c = ab$ for three elements of A. Then a is called a *left divisor* of c and b is called a *right divisor* of c.

Definition 131 (Commutative subset)**.** A subset S of A is called a *commutative subset* if for each $s, t \in S$ we have $st = ts$.

Definition 132 (Subtractive subset)**.** We borrow the terminology of [NSM00, Definition 2.2 page 3] and say that a subset S of A is *subtractive* if for all $s, t \in S$ with $s \neq t$ we have $s - t \in A^\times$.

Let $A[X]$ be the ring of polynomials over A and, for a *positive* integer k let $A[X]_{<k}$ be the bimodule of all polynomials of $A[X]$ of degree at most $k - 1$. Then $A[X]$ is commutative if and only if A is. We denote by $Z(A)$ the center of A. We have $Z(A[X]) = Z(A)[X]$.

Definition 133 (Evaluation map)**.** Let

$$f = \sum_0^l f_i X^i \in A[X]$$

and $a \in A$. We define the *evaluation* of f at a, denoted by $f(a)$, to be the element

$$\sum_0^l f_i a^i \in A.$$

In general, the evaluation map $f \mapsto f(a)$ is not a ring homomorphism. Note however that $f \mapsto f(a)$ is a ring homomorphism whenever $a \in Z(A)$. Let $g \in A[X]$ and suppose that the subset of A constituted by the coefficients of g and a is commutative. Then we have $(fg)(a) = f(a)g(a)$. We call a a *root* of f if $f(a) = 0$.

Let $f \in A[X]$ and $x = (x_1, \ldots, x_n) \in A^n$. For convenience sake the vector $(f(x_1), f(x_2), \ldots, f(x_n))$ of A^n will be denoted by $f(x)$. We include in this section propositions about evaluation and interpolation of polynomials of $A[X]$.

Lemma 134. *Let f and g be nonzero polynomials over A such that the leading coefficient of g is a unit of A. Then there exist unique polynomials $q_l(X), r_l(X) \in A[X]$ such that $f = q_l g + r_l$ and $\deg r_l < \deg g$; and unique polynomials $q_r(X), r_r(X) \in A[X]$ such that $f = g q_r + r_r$ and $\deg r_r < \deg g$.*

Remark 135. Taking the same notations as in Lemma 134 note that we have $q_l = q_r$ and $r_l = r_r$ whenever the coefficients of f and of q_r *or* q_l form a commutative subset of A. It is the case in particular when $g \in Z(A[X])$. A direct consequence of Lemma 134 is that $a \in A$ is a root of f if and only if $X - a$ is a right divisor of f. Moreover if $a \in Z(A)$ then a is root of f if and only if $X - a$ is a right and left divisor of f. The following corollary is the key ingredient of many proofs in this paper.

Corollary 136. *Let f be a polynomial over A of degree at most n with at least $n+1$ roots contained in a commutative subtractive subset of A. Then $f = 0$. It is the case in particular when the considered $(n+1)$ roots are in $Z(A)$.*

The next corollary of Lemma 134 allows to do Lagrange interpolation as soon as the points at which interpolation is done are well chosen.

Corollary 137. *Let $\{x_1, \ldots, x_{n+1}\}$ be a commutative subtractive subset of A and $\{y_1, \ldots, y_{n+1}\}$ be a subset of A. Then there exists a unique polynomial $f \in A[X]$ of degree at most n such that $f(x_i) = y_i$, for $i = 1, \ldots, n+1$.*

Proof. The proof is included to introduce some notations that will be useful later.

The uniqueness is a direct consequence of Corollary 136. Let

$$L_i(X) = \prod_{j \neq i} (X - x_j).$$

Note that if $h \in A[X]$ and $\lambda, x \in A$ then $(\lambda h)(x) = \lambda(h(x))$. Therefore the polynomial

$$f(X) = \sum_{i=1}^{n+1} y_i L_i(x_i)^{-1} L_i(X), \tag{3.1}$$

verifies $f(x_i) = y_i$ for $i = 1, \ldots, n+1$. □

As in the commutative case, we will need to work in a localization of A. However the operation of localization is slightly more complicated. Let S be a multiplicative subset of A *i.e.* S satisfies $s, t \in S \Rightarrow st \in S$. Following [MRS01, Paragraph 1.6, page 43] one can form the ring of right fractions denoted by AS^{-1} under certain conditions such as the *right Ore condition*.

Definition 138 (Right Ore condition)**.** A subset S of A is said to satisfy the right Ore condition if for all $r \in A$ and $s \in S$ there exists $r' \in A$ and $s' \in S$ such that $rs' = sr'$.

Following [MRS01, Definition 1.3, page 42], we denote by $\mathrm{ass}(S)$ the set $\{a \in A : as = 0 \text{ for some } s \in S\}$.

Proposition 139. *Let S be a multiplicative subset of A satisfying the right Ore condition such that the image of S in the quotient ring $A/\mathrm{ass}(S)$ consists of regular elements of $A/\mathrm{ass}(S)$. Then the ring of right fractions of A with respect to S exists. We will denote it by AS^{-1}.*

Proof. See [MRS01, Theorem 1.12, page 47]. □

3.2.1 Error correcting codes

As in the case of linear error correcting codes over a finite field, we define a linear error correcting code as a submodule of A^n for a positive integer n. But, as A is not commutative *a priori* we have to define *left* and *right* linear error correcting codes.

Definition 140 (Left and right linear code)**.** Let n be a positive integer. A *left* (resp. *right*) *linear error correcting code* is a left (resp. right) submodule C of A such that for each $i \in \{1, \ldots, n\}$ there exists an element of C such that its i'th coordinate is nonzero.

The elements of C are called *codewords* while the elements of A^n are called *words*.

If C is a bi submodule of A^n then it is called a *linear error correcting code*, or simply an *error correcting code* or a *code* if there is no confusion on the linearity of C.

As in the finite field case, we can define the Hamming distance and weight.

Definition 141 (Hamming weight and distance)**.** Let $u \in A^n$. We call the *Hamming weight* of u the number of nonzero entries of u and denote this number by $w(u)$. Now let $v \in A^n$. The *Hamming distance* between u and v is the nonnegative integer $w(u-v)$ and denoted by $d_H(u, v)$. If there is no confusion the Hamming distance between u and v will be simply called the *distance* between u and v and denoted by $d(u, v)$.

Let C be a *subset* of A^n. The *minimum Hamming distance* of C denoted by d_C is defined as

$$d_C = \min_{u,v \in C \text{ and } u \neq v} d(u, v).$$

If there is no confusion the minimum Hamming distance of C is simply called the *minimum distance* of C. Note that if C is an additive subgroup of A^n we have

$$d_C = \min_{u \in C \setminus \{0\}} w(u).$$

Definition 142 (Support)**.** Let $u = (u_1, \ldots, u_n) \in A^n$. The set

$$\{i \in \{1, \ldots, n\} : u_i \neq 0\}$$

is called the *support* of u and denoted by $\mathrm{Supp}(u)$.

Definition 143 (Inner product)**.** Let $u = (u_1, \ldots, u_n)$ and $v = (v_1, \ldots, v_n)$ be two elements of A^n. The *inner product* of u and v is defined by

$$\langle u, v \rangle = \sum_{i=1}^{n} u_i v_i \in A.$$

Let C be a bi submodule of A^n. We can define the *left dual* submodule ${}^{\perp}C$ of A^n to be the left submodule of A^n defined by

$${}^{\perp}C = \{u \in A^n : \forall c \in C,\ \langle u, c \rangle = 0\},$$

and similarly the *right dual* submodule $C^{\perp}$ of A^n to be the right submodule of A^n defined by

$$C^{\perp} = \{u \in A^n : \forall c \in C,\ \langle c, u \rangle = 0\}.$$

A priori there is no reason that ${}^{\perp}C = C^{\perp}$ except under special hypothesis.

Definition 144 (Parity-check matrix)**.** Let C be a code such that ${}^{\perp}C = C^{\perp}$. Suppose moreover that $C^{\perp}$ has a base $(b_1, \ldots, b_s)$ where b_i is a row matrix for $i = 1, \ldots, s$. Then the matrix

$$\begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_s \end{pmatrix}$$

is called a *parity-check* matrix of C.

3.2.2 Galois rings

We recall briefly basic results about Galois rings that will be useful throughout the article. We fix for this subsection a prime number p and two positive integers r and s.

Proposition 145. *Let $\varphi(X), \psi(X) \in (\mathbb{Z}/p^r\mathbb{Z})[X]$ be monic polynomials of degree s, irreducible modulo p. Then we have a ring isomorphism:*

$$\frac{(\mathbb{Z}/p^r\mathbb{Z})[X]}{(\varphi(X))} = \frac{(\mathbb{Z}/p^r\mathbb{Z})[X]}{(\psi(X))}.$$

Proof. See [Rag69, Statements I and II, page 207]. □

Definition 146 (Galois ring)**.** With the same notation as Proposition 145, the ring

$$\frac{(\mathbb{Z}/p^r\mathbb{Z})[X]}{(\varphi(X))}$$

is denoted by $\mathrm{GR}(p^r, s)$ and called a *Galois ring.*

Proposition 147. *The Galois ring $A = \mathrm{GR}(p^r, s)$ is a finite local ring whose maximal ideal is generated by p. Its residue field is $\mathbb{F}_{p^s}$. Moreover all the ideals of A are principal and generated by a power of p.*

Proof. The proposition follows from [Rag69, Paragraph 3.5, page 212] and [Ser62, Theorem 7, page 52]. □

In order to use Galois rings as a suitable alphabet for our decoding algorithms we need the following proposition.

Proposition 148. *Let $\mathbb{Z}_p$ be the ring of p-adic integers. We let $\mathbb{Z}_{p^s}$ denote an unramified extension of $\mathbb{Z}_p$ of degree s. Then $\mathrm{GR}(p^r, s)$ and $\mathbb{Z}_{p^s}/(p^r)$ are isomorphic as rings.*

Proof. The proposition follows from [Rag69, Paragraph 3.5, page 212] and [Ser62, Theorem 7, page 52]. □

3.2.3 Complexity model

In order to analyze the performances of our algorithms, we let $\mathsf{I}(n)$ be the time needed to multiply two integers of bit-size at most n in *binary representation*. It is classical ([CK91], [Für07], [SS71]) that we can take $\mathsf{I}(n) \in O(n \log n 2^{\log^* n})$, where $\log^*$ represents the iterated logarithm of n. If A is a commutative ring, we let $\mathsf{M}_A(n)$ be the cost of multiplying two polynomials of degree at most n with coefficients in A in terms of the number of arithmetic operations in A. It is well known ([GG03, Theorem 8.23, page 240]) that we can take $\mathsf{M}_A(n) \in \tilde{O}(n)$. Thus the bit-cost of multiplying two elements of $\mathbb{F}_{p^n}$ is $\tilde{O}(n \log p)$ where p is a prime number.

Finally, let us recall that the *expected cost* spent by a randomized algorithm is defined as the average cost for a given input over all the possible executions.

3.3 Generalized Reed-Solomon codes

In this section, we extend the main propositions about GRS codes over a ring. We study their parameters, duality, key equation, weight distribution and the MacWilliams identity.

From now on and until the end of this article we fix three positive integers $k < n$ and $d = n - k + 1$, a commutative subtractive subset $\{x_1, \ldots, x_n\}$ of A, $x = (x_1, \ldots, x_n)$ and $v = (v_1, \ldots, v_n) \in (Z(A)^\times)^n$.

Definition 149 (Generalized Reed-Solomon code)**.** The left submodule of A^n generated by the vectors of the form

$$(v_1 f(x_1), \ldots, v_n f(x_n)) \in A^n,$$

with $f \in A[X]_{<k}$ is denoted by

$$\mathrm{GRS}_A(v, x, k) = \mathrm{GRS}_A((v_1, \ldots, v_n), (x_1, \ldots, x_n), k)$$

and is called the *generalized Reed-Solomon* code over A of parameters $[v, x, k]$ or simply $[n, k]$ if there is no confusion on v and x. The integer n is called the *code block length* or simply *length* of $\mathrm{GRS}_A(v, x, k)$. The n-tuple $v = (v_1, \ldots, v_n)$ is called the *weight* of $\mathrm{GRS}_A(v, x, k)$. The n-tuple $x = (x_1, \ldots, x_n)$ is called the *support* of the code. When there is no confusion on the ring A, the weight and the support, we will simply write $\mathrm{GRS}(n, k)$ for $\mathrm{GRS}_A(v, x, k)$. The integer k will be called the *pseudo-dimension* of $\mathrm{GRS}(n, k)$ throughout this paper. When $v = (1_A, \ldots, 1_A)$ we call $\mathrm{GRS}_A(v, x, k)$ a Reed-Solomon code and denote it by $\mathrm{RS}_A(v, x, k)$ or simply $\mathrm{RS}(n, k)$ if there is no confusion on the ring A, the weight and the support.

Note that if $\{x_1, \ldots, x_n\} \subseteq Z(A)$ then the left linear code $\mathrm{GRS}_A(v, x, k)$ defined in Definition 149 is in fact a code (*i.e.* a bi submodule of A^n).

Proposition 150. *The left linear code $\mathrm{GRS}(n, k)$ is free and has a left basis of cardinality k. In particular when A is commutative the pseudo-dimension k of $\mathrm{GRS}(n, k)$ corresponds to its rank.*

Proof. Let

$$\begin{array}{rccc} \mathrm{ev} : & A[X]_{<k} & \longrightarrow & A^n \\ & f(X) & \longmapsto & (v_1 f(x_1), \ldots, v_n f(x_n)). \end{array}$$

Suppose that $\mathrm{ev}(f) = 0$, then $f(x_i) = 0$ for $i = 1, \ldots, n$. Therefore by Corollary 136 we must have $f = 0$ and ev is injective. But $A[X]_{<k}$ is free and has a basis of cardinality k namely $(1, X, X^2, \ldots, X^{k-1})$ hence the proposition. □

Corollary 151. *The code* $\mathrm{GRS}(n, k)$ *has minimum distance* $d = n - k + 1$.

Proof. Denote by d' the minimum distance of $\mathrm{GRS}(n, k)$. Let $g = (X - x_1)(X - x_2) \cdots (X - x_{k-1})$ be of degree $k - 1$. As $\{x_1, \ldots, x_n\}$ is a commutative subset of A, we have $g(x_i) = 0$ for $i = 1, \ldots, k - 1$ and $g(x_i)$ is a product of units of A by hypothesis, thus $g(x_i) \neq 0$ for $i = k, \ldots, n$. Therefore $d' \leq n - k + 1$.

Suppose now that there exists a polynomial $f \in A[X]_{<k}$ such that $f(x_i) = 0$ for at least k values of i. By Corollary 136 we must have $f = 0$. Thus $d' \geq n - k + 1$. □

Proposition 152. *A generator matrix of* $\mathrm{GRS}_A(v, x, k)$ *is given by*

$$\begin{pmatrix} v_1 & v_2 & \ldots & v_n \\ v_1 x_1 & v_2 x_2 & \ldots & v_n x_n \\ v_1 x_1^2 & v_2 x_2^2 & \ldots & v_n x_n^2 \\ \ldots & \ldots & \ldots & \ldots \\ v_1 x_1^{k-1} & v_2 x_2^{k-1} & \ldots & v_n x_n^{k-1} \end{pmatrix}.$$

Proposition 153. *Let* S *be a commutative subtractive subset of* A. *Then there exists a* commutative *ring* B *such that* $|B| = |A|$, *and a subtractive subset* T *of* B *such that* $|T| = |S|$.

Proof. Let $m = |A|$. We write

$$m = \prod_{i=1}^{r} p_i^{l_i},$$

where $l_1, \ldots, l_r$ are positive integers and $p_1, \ldots, p_r$ are prime numbers. We let $Z = Z(A)[S]$ the commutative subring of A generated by the elements of S over $Z(A)$.

We first prove that p_i divides $|Z|$ for $i = 1, \ldots, r$. Suppose that it is not the case, say p_1 does not divide $|Z|$. Then p_1 divides the order of the quotient additive group A/Z. Then by [Lan02, Lemma 6.1 page 33] there exists $a \in A \setminus Z$ such that $p_1 a \in Z$. But $p_1 1_A = p_1 1_Z$ is invertible in Z and we have $a = (p_1 1_Z)^{-1}(p_1 1_Z)a = (p_1 1_Z)^{-1}(p_1 a) \in Z$.

Now Z is a finite commutative ring and we can write by [AM94, Theorem 8.7, page 90]

$$Z = \prod_{i=1}^{s} \mathfrak{A}_i$$

in a unique way (up to isomorphism) where $\mathfrak{A}_i$ is a finite local commutative ring for $i = 1, \ldots, s$. We must have $s \geq r$: by [Rag69, Theorem 2, page 199] the cardinality of

$\mathfrak{A}_i$ is a prime power, thus if $s < r$ it would contradict the fact that p_i divides $|Z|$ for $i = 1, \ldots, r$.

Denote by $\mathfrak{m}_i$ the maximal ideal of $\mathfrak{A}_i$ for $i = 1, \ldots, r$. Since S is a subtractive subset of Z we must have

$$|S| \leq \min_{i \in \{1,\ldots,r\}} |\mathfrak{A}_i/\mathfrak{m}_i| \,. \tag{3.2}$$

We can assume that the right hand side of the above inequality is equal to $|\mathfrak{A}_1/\mathfrak{m}_1|$ without loss of generality. There exists a positive integer l such that $|\mathfrak{A}_1/\mathfrak{m}_1| = p_1^l$ and we have $|S| \leq p_1^l \leq |\mathfrak{A}_i| \leq p_i^{l_i}$ for $j = 2, \ldots, r$ and $j = 1$.

Now let $B = \prod_{i=1}^{s} \mathbb{F}_{p_i^{l_i}}$ be the product ring of finite fields. Then Inequality 3.2 implies that B contains a subtractive subset T such that $|T| = |S|$. □

We have proved Theorem 128.

Theorem 154. *For a GRS code over a finite ring A with parameters $[n, k, n-k+1]_A$, there exists a GRS code over a commutative ring B with $|B| = |A|$ and of parameters $[n, k, n-k+1]_B$.*

Proof. It is a direct consequence of Proposition 153. □

Theorem 155. *Suppose that A is finite and that $\{x_1, \ldots, x_n\} \subseteq Z(A)$. Then ${}^{\perp}\mathrm{GRS}_A(v, x, k) = \mathrm{GRS}_A(v, x, k)^{\perp} = \mathrm{GRS}_A(v', x, n-k)$ where $v' = (v'_1, \ldots, v'_n)$ with*

$$v'_i = \left(v_i \prod_{j \neq i} (x_i - x_j) \right)^{-1} \in Z(A).$$

Proof. In short: we let $C = \mathrm{GRS}_A(v, x, k)$, $b_i = (v_1 x_1^i, v_2 x_2^i, \ldots, v_n x_n^i) \in Z(A)^n$ for $i = 0, \ldots, k-1$ and $C' = \mathrm{GRS}_A(v', x, n-k)$.

We first prove that $C' \subseteq C^{\perp}$. Let $f(X) \in A[X]_{<k}$ and $g(X) \in A[X]_{<n-k}$. Then $fg \in A[X]_{<n-1}$. According to Equation 3.1 we have

$$f(X)g(X) = \sum_{i=1}^{n} f(x_i)g(x_i)L_i(x_i)^{-1}L_i(X).$$

Equating coefficients of degree $n-1$, we get

$$0 = \sum_{i=1}^{n} v_i f(x_i) \left(v_i L_i(x_i)\right)^{-1} g(x_i).$$

Hence $C' \subseteq C^{\perp}$. Doing the same with gf instead of fg, we also get $C' \subseteq {}^{\perp}C$.

We now have to prove that $C' = C^{\perp}$. Let $c' = (c'_1, \ldots, c'_n) \in A^n$. Then $c' \in C^{\perp}$ if and only if

$$\begin{pmatrix} v_1 & v_2 & \ldots & v_n \\ v_1 x_1 & v_2 x_2 & \ldots & v_n x_n \\ v_1 x_1^2 & v_2 x_2^2 & \ldots & v_n x_n^2 \\ \ldots & \ldots & \ldots & \ldots \\ v_1 x_1^{k-1} & v_2 x_2^{k-1} & \ldots & v_n x_n^{k-1} \end{pmatrix} \begin{pmatrix} c'_1 \\ c'_2 \\ \vdots \\ c'_n \end{pmatrix} = 0. \tag{3.3}$$

The matrix has its coefficients in the commutative ring $Z(A)$ thus we can compute the determinant of

$$V = \begin{pmatrix} v_1 & v_2 & \dots & v_k \\ v_1x_1 & v_2x_2 & \dots & v_kx_k \\ v_1x_1^2 & v_2x_2^2 & \dots & v_kx_k^2 \\ \dots & \dots & \dots & \dots \\ v_1x_1^{k-1} & v_2x_2^{k-1} & \dots & v_kx_k^{k-1} \end{pmatrix}.$$

And we have

$$\det V = \left(\prod_{i=1}^{k} v_i\right)\left(\prod_{i\neq j}(x_i - x_j)\right).$$

This determinant is a unit of $Z(A)$ by the hypothesis made on the weight v and the support x of the code C. Thus V has an inverse in $M_k(Z(A))$ and therefore V^{-1} is also the left and right inverse of V in $M_k(A)$. As a consequence given $(c'_{k+1}, \dots, c'_n) \in A^{n-k}$ there exists one and only one k-tuple $(c'_1, \dots, c'_k) \in A^k$ such that Equation 3.3 is satisfied. Thus the number of solutions in A^n of Equation 3.3 is equal to $|A|^{n-k}$ which is also the number of elements of C'.

Using the fact that $(b_i)_{i=0,\dots,k-1}$ is a basis of C as a right module and the following system of equations

$$\left(c'_1, c'_2, \dots, c'_n\right)\begin{pmatrix} v_1 & v_1x_1 & \dots & v_1x_1^{k-1} \\ v_2 & v_2x_2 & \dots & v_2x_2^{k-1} \\ \dots & \dots & \dots & \dots \\ v_n & v_nx_n & \dots & v_nx_n^{k-1} \end{pmatrix} = 0,$$

(which is the transposed system of system 3.3) instead of the system 3.3 of equations, we get $C' = {}^{\perp}C$. □

Corollary 156. *Suppose that A is finite and that $\{x_1, \dots, x_n\} \subseteq Z(A)$. Then there exists a parity-check matrix for $\mathrm{GRS}_A(v, x, k)$ equal to*

$$\begin{pmatrix} v'_1 & v'_2 & \dots & v'_n \\ v'_1x_1 & v'_2x_2 & \dots & v'_nx_n \\ v'_1x_1^2 & v'_2x_2^2 & \dots & v'_nx_n^2 \\ \dots & \dots & \dots & \dots \\ v'_1x_1^{n-k-1} & v'_2x_2^{n-k-1} & \dots & v'_nx_n^{n-k-1} \end{pmatrix}$$

where the v'_i are defined as in Theorem 155.

Lemma 157 (Goppa formulation for GRS codes)**.** *When A is finite and $\{x_1, \dots, x_n\} \subseteq Z(A)$, we have, for $c = (c_1, \dots, c_n) \in A^n$,*

$$c \in \mathrm{GRS}_A(v, x, k) \iff \sum_{i=1}^{n} \frac{c_iv'_i}{1 - x_iX} = 0 \mod X^{n-k} \tag{3.4}$$

where the v'_i are defined as in Theorem 155.

Proof. We check that equation 3.4 makes sense. First, The ideal generated by X^{n-k} is a two sided ideal. Secondly, in $A[[X]]$ the power series $(1 - f(X))$ where $f(X) \in XA[[X]]$ has a two-sided inverse given by $\sum_{i=0}^{\infty} f(X)^i X^i$. This can be shown by direct computation.

The rest of the proof is also a direct computation and is left to the reader. □

Proposition 158 (Key equation for GRS codes)**.** *Suppose that A is finite and that $\{x_1, \ldots, x_n\} \subseteq Z(A)$. Let $y = (y_1, \ldots, y_n) \in A^n$ be the received word such that there exists a unique codeword $c = (c_1, \ldots, c_n) \in \mathrm{GRS}(n, k)$ at distance at most $\left\lfloor \frac{n-k}{2} \right\rfloor$. We let $e = (e_1, \ldots, e_n) = y - c$ and $E = \mathrm{Supp}(e)$. Let*

$$\sigma(X) = \prod_{i \in E} (1 - x_i X),$$

$$\omega(X) = \sum_{i \in E} e_i v_i' \left(\prod_{j \in E \text{ and } j \neq i} (1 - x_j X) \right)$$

and

$$S(X) = \sum_{i=1}^{n} \frac{y_i v_i'}{1 - x_i X} \mod X^{n-k}.$$

Then we have $\sigma(X)S(X) = \omega(X) \mod X^{n-k}$.

Proof. This is a direct computation using Lemma 157. □

The following proposition will be useful to compute the complexities of the proposed algorithms and to prove MacWilliams identity for GRS codes over non commutative rings.

Proposition 159. *Suppose that A is finite. Let $\mathcal{C}_s$ be the number of codewords from $\mathrm{GRS}(n, k)$ of weight s. Then $\mathcal{C}_0 = 1$, $\mathcal{C}_s = 0$ for $1 \leq s \leq d - 1$ and*

$$\mathcal{C}_s = \binom{n}{s} \sum_{i=0}^{s-n+k} (-1)^i \binom{s}{i} \left(|A|^{s-n+k+1-i} - 1 \right) \tag{3.5}$$

for $d \leq s \leq n$.

Proof. The result is obvious for $\mathcal{C}_0$ and $\mathcal{C}_s$ for $1 \leq s \leq d - 1$. So now let $d \leq s \leq n$ and denote by $N(i_1, \ldots, i_s)$ the number of codewords with zeros at coordinates $i_1, \ldots, i_s$. Then $|N(i_1, \ldots, i_s)| = |A|^{k-s}$ by Remark 135 and Corollary 136. The rest of the proof is identical to [PW72, paragraph 3.9, page 79]. □

Proposition 160. *Suppose that A is finite and that $\{x_1, \ldots, x_n\} \subseteq Z(A)$. With the notations of Proposition 159, let $\mathcal{C} = \mathrm{GRS}_A(v, x, k)$,*

$$W_{\mathcal{C}}(X, Y) = \sum_{i=0}^{n} \mathcal{C}_i X^{n-i} Y^i,$$

and

$$W_{\mathcal{C}^\perp}(X,Y) = \sum_{i=0}^{n} (\mathcal{C}^\perp)_i X^{n-i} Y^i.$$

Then

$$W_{\mathcal{C}}(X,Y) = \frac{1}{|A|^{n-k}} W_{\mathcal{C}^\perp}(X + (|A|-1)Y, X - Y).$$

Proof. By Proposition 153 there exists a RS code over a commutative ring B of cardinality $|A|$. We denote by $\mathcal{D}$ this code. It has parameters $[n, k, n-k+1]_B$ over B. We can take B to be a product of finite fields by 153 again. Seen as module over itself, B is semisimple and thus by [Smi81, Paragraph 1, page 989] is injective. Now by [Woo99, Theorem 1.2 page 4] and [Woo99, Remark 1.3, page 4] B is a Frobenius ring and [Woo99, Theorem 8.3 page 18] can be applied. By Proposition 159 the weight distribution of $\mathcal{C}$ is the same as the one of $\mathcal{D}$.

Gathering the above results we finally get

$$\begin{aligned} W_{\mathcal{C}}(X,Y) &= W_{\mathcal{D}}(X,Y) \\ &= \frac{1}{|B|^{n-k}} W_{\mathcal{D}^\perp}(X + (|B|-1)Y, X - Y) \\ &= \frac{1}{|A|^{n-k}} W_{\mathcal{C}^\perp}(X + (|A|-1)Y, X - Y). \end{aligned}$$

□

3.4 Unique decoding of generalized Reed-Solomon codes

3.4.1 Unique decoding over certain valuation rings

In this section we design a unique decoding algorithm for GRS codes over a discrete valuation ring. We suppose that A has the following property:

(∗) *there exists a regular element p which is not a unit such that $p \in Z(A)$ and such that every element $a \in A$ can be uniquely written as $\sum_{i=0}^{\infty} a_i p^i$ where, for all $i \in \mathbb{N}$, a_i is in a set of representatives of $A/(p)$.*

It is the case for example for the two rings $\mathbb{Z}_q$ and $M_\ell(\mathbb{Z}_q)$ where $\mathbb{Z}_q$ denotes an unramified extension of the p-adic numbers or for the power series ring $\kappa[[t]]$ and the ring of matrices $M_\ell(\kappa[[t]])$. We will need in this section and the rest of this paper to divide elements of A by p, which is provided by the following lemma.

Lemma 161. *The ring of right fractions with respect to the subset S of A formed by the powers of p exists.*

Proof. Clearly S is a multiplicative subset of A. The fact that p is in the center of A and regular implies that S satisfies the right Ore condition (Definition 138) and that $\mathrm{ass}(S) = \{0\}$. Therefore we can apply Proposition 139. □

Lemma 161 in combination with [MRS01, Paragraph 1.3 (iii), page 42] shows that we have a natural injection $A \subseteq AS^{-1}$ and we will freely use this injection to identify elements of A and their images in the ring AS^{-1}.

We let $\Lambda = A/(p)$ be the residual ring of A. In the sequel for a vector $b = (b_1, \ldots, b_n) \in A^n$, we will denote by $b \mod p$ the vector $(b_1 \mod p, \ldots, b_n \mod p) \in (A/(p))^n$ and if $f \in A[X]$, recall that we denote by $f(b)$ the vector $(f(b_1), \ldots, f(b_n)) \in A^n$.

Let $\mathcal{C} = \mathrm{GRS}_A(v, x, k)$ be a generalized Reed-Solomon code. Then $\mathcal{C}/p^r\mathcal{C} = \mathrm{GRS}_{A/(p^r)}(v \mod p^r, x \mod p^r, k)$.

Lemma 162. *Let $c \in \mathcal{C}$ such that $c/p \in A^n$. Then $c/p \in \mathcal{C}$.*

Proof. Let $f \in A[X]_{<k}$ such that $c = f(x)$. Then $f(x) \mod p = 0$. By Corollary 136 we have $f \mod p = 0$ and there exists $g \in A[X]_{<k}$ such that $f(X) = pg(X)$. □

We now give an algorithm of unique decoding for GRS codes over $B = A/(p^r)$ for a positive integer r. We let $\tau = \left\lfloor \frac{n-k}{2} \right\rfloor$. The idea of the algorithm is to do a Hensel lifting. We first look at the received word modulo p. Then we call a decoding algorithm for the GRS code modulo p. It then returns the component of degree 0 of the wanted codeword and the error. We subtract these to the received word and then can divide the result by p to reiterate the process and obtain the component of degree 1, 2 up to $r-1$. We first precise what is the black box algorithm.

Algorithm 15 Black box unique decoding algorithm

Input: a received vector y of Λ^n with at most $\left\lfloor \frac{n-k}{2} \right\rfloor$ errors.
Output: the message $m \in \Lambda^k$ such that the corresponding codeword is within distance $\left\lfloor \frac{n-k}{2} \right\rfloor$ of y and the error $e \in \Lambda^n$.

In the following algorithm we denote by G a generator matrix of $\mathcal{C}$.

Algorithm 16 Unique decoding over a valuation ring

Input: a received vector $y = (y_1, \ldots, y_n) \in B^n$ with at most τ errors, and a **black box** unique decoding algorithm for $\mathcal{C}/p\mathcal{C}$ as Algorithm 15.
Output: the unique codeword of $\mathcal{C}/p^r\mathcal{C}$ within distance τ of y.

1: Compute $y_0 \in A$ such that $y_0 \mod p^r = y$.
2: **for** $i = 0 \to r-1$ **do**
3: Call the **black box** with input $y_i \mod p$ and obtain $m_i \in \Lambda^k$ and $e_i' \in \Lambda^n$.
4: $c_i \leftarrow m_i G \in \mathcal{C}$.
5: $e_i \leftarrow$ a representative of e_i' such that $\mathrm{Supp}(e_i) = \mathrm{Supp}(e')$.
6: $y_{i+1} \leftarrow (y_i - c_i - e_i)/p$.
7: **end for**
8: **return** $\sum_{i=0}^{r-1} p^i c_i \mod p^r$.

Proposition 163. *Algorithm 16 is correct and can decode up to τ errors.*

Proof. We let $c \in \mathcal{C}$ and $e \in A^n$ be such that $w(e) \leq \tau$ and $y = c + e$. We will show by induction on i that the following items holds after step 7 of Algorithm 16:

1. $y_0 = \sum_{j=0}^{i} p^j(c_j + e_j) + p^{i+1}y_{i+1}$,
2. $y_{i+1} = (y_i - c_i - e_i)/p = c'' + e''$ where $c'' \in \mathcal{C}$ and $e'' \in A^n$ with $\mathrm{Supp}(e'') \subseteq \mathrm{Supp}(e)$.

For $i = 0$ item 1 is trivially satisfied and we have at step 4

$$y \mod p = (c + e) \mod p = c' + e'$$

where $c' \in \mathcal{C}/p\mathcal{C}$ and $e' \in \Lambda^n$ such that $w(e') \leq \tau$. By unicity of c' we must have $c \mod p = c'$ and $e \mod p = e'$. Therefore we have after step 6

$$(y_0 - c_0 - e_0)/p = (c - c_0)/p + (e - e_0)/p$$

and by Lemma 162 we have $(c - c_0)/p \in \mathcal{C}$. Moreover $\mathrm{Supp}(e_0) \subseteq \mathrm{Supp}(e)$ thus item 2 is satisfied.

Now suppose that the induction holds for $i \geq 0$. Then we get from item 2 of the inductive hypothesis and from step 4

$$y_{i+1} \mod p = (c'' + e'') \mod p = c' + e'$$

where $c'' \in \mathcal{C}$ and $e'' \in A^n$ such that $\mathrm{Supp}(e'') \subseteq \mathrm{Supp}(e)$, thus $w(e'') \leq \tau$. Therefore by unicity of c' we must have $c'' \mod p = c'$ and $e'' \mod p = e'$. Therefore we deduce

$$(y_{i+1} - c_{i+1} - e_{i+1})/p = (c'' - c_{i+1})/p + (e'' - e_{i+1})/p$$

and by Lemma 162 $(c'' - c_{i+1})/p \in \mathcal{C}$. Moreover $\mathrm{Supp}(e_{i+1}) \subseteq \mathrm{Supp}(e'') \subseteq \mathrm{Supp}(e)$ and item 2 is satisfied. As for item 1, we have

$$\begin{aligned} p^{i+2}y_{i+2} &= p^{i+1}(y_{i+1} - c_{i+1} - e_{i+1}) \\ &= y_0 - \sum_{j=0}^{i} p^j(c_j + e_j) - p^{i+1}(c_{i+1} + e_{i+1}). \end{aligned}$$

Now taking $i = r$, we get

$$y = y_0 \mod p^r = \sum_{j=0}^{r-1} p^j c_j + \sum_{j=0}^{r-1} p^j e_j \mod p^r.$$

We have $c_j \in \mathcal{C}$ for $j = 1, \ldots, r-1$ thus

$$\sum_{j=0}^{r-1} p^j c_j \mod p^r \in \mathcal{C}/p^r\mathcal{C}.$$

As $\mathrm{Supp}(e_j) \subseteq \mathrm{Supp}(e)$ for $j = 1, \ldots, r-1$ we have

$$w\left(\sum_{j=0}^{r-1} p^j e_j \mod p^r\right) \leq \tau.$$

Therefore the unicity of c implies that

$$c = \sum_{j=0}^{r-1} p^j c_j \mod p^r.$$

□

Proposition 164. *Let* $\mathsf{UDec}(\mathcal{C})$ *be the complexity of the black box decoding algorithm for* $\mathcal{C}/p\mathcal{C}$ *in terms of the number of bit-operations given as input of Algorithm 16 and* $\mathsf{Lift}(\mathcal{C})$ *the complexity of lifting a codeword from* $\mathcal{C}/p\mathcal{C}$ *into* $\mathcal{C}$ *(*i.e. *the bit-cost of step 4 of Algorithm 16). Then Algorithm 16 performs a number of* $r(\mathsf{UDec}(\mathcal{C}) + \mathsf{Lift}(\mathcal{C}))$ *bit-operations.*

Lemma 165. *Suppose that* $v = (1_B, \ldots, 1_B)$*. If* $B = \mathrm{GR}(p^r, s)$ *we can take* $\mathsf{Lift}(\mathcal{C}) = O(nk\,\mathsf{M}_{\mathbb{F}_p}(s)\,\mathsf{I}(\log p))$ *bit-operations. If* $B = \mathbb{F}_{p^s}[[t]]/(t^r)$ *we can take* $\mathsf{Lift}(\mathcal{C}) = O(nk)$ *arithmetic operations in* $\mathbb{F}_{p^s}$*; in the situation where the support of* $\mathcal{C}$ *is contained in* $\mathbb{F}_{p^s}$ *we can take* $\mathsf{Lift}(\mathcal{C}) = O(M_{\mathbb{F}_{p^s}}(n)\log n)$ *arithmetic operations in* $\mathbb{F}_{p^s}$*.*

Proof. Lifting a codeword from $\mathcal{C}/p\mathcal{C}$ into $\mathcal{C}$ can be done by the matrix-vector product mG where G is a generator matrix of $\mathcal{C}$ and

- $m \in B^k$ is a representative of the message modulo p whose coefficients have the same bit-size as elements of $\mathbb{F}_p$ when $B = \mathrm{GR}(p^r, s)$,
- $m \in \mathbb{F}_{p^s}$ is a representative of the message modulo t when $B = \mathbb{F}_{p^s}[[t]]/(t^r)$.

In the situation where $B = \mathbb{F}_{p^s}[[t]]/(t^r)$ and that the support of $\mathcal{C}$ is included in $\mathbb{F}_{p^s}$ we can use fast multipoint evaluation of a polynomial of degree at most n with coefficients in $\mathbb{F}_{p^s}$ in n points of $\mathbb{F}_{p^s}$ which is done in $O(M_{F_{p^s}}(n)\log n)$ by [GG03, Corollary 10.8, page 295]. □

Corollary 166. *Suppose that* B *is the ring* $\mathbb{F}_{p^s}[[t]]/(t^r)$*. Then there exists a unique decoding algorithm for* $RS_A(v, x, k)$ *where* $x \in \mathbb{F}_{p^s}$ *with an asymptotic complexity of* $\tilde{O}(rn)$ *arithmetic operations in* $\mathbb{F}_{p^s}$*.*

Proof. This is a direct consequence of Proposition 164, Lemma 165 and [Jus76]. □

Remark 167. Taking the notations as Corollary 166 note that, given any GRS code C over B, we can always find a GRS code C' over B with same parameters as C such that its support is included in $\mathbb{F}_{p^s}$.

We denote by $\mathfrak{D}$ the unique decoding algorithm that can be found in [Ber68, Mas69, SKHN75, Jus76, BW86, Gao02].

Theorem 168. *Given a finite field A, a truncated power series ring B such that $|A| = |B|$, a RS code $\mathcal{C}_A$ over A of parameters $[n, k, n-k+1]_A$ and a unique decoding algorithm* `UDec` *from the list $\mathfrak{D}$ for $\mathcal{C}_A$. Suppose that there exists a RS code $\mathcal{C}_B$ over B of parameters $[n, k, n-k+1]_B$. Then there exists a RS code $\mathcal{C}'_B$ over B of parameters $[n, k, n-k+1]_B$ such that $\mathcal{C}_B/p\mathcal{C}_B = \mathcal{C}'_B/p\mathcal{C}'_B$ and a unique decoding algorithm for $\mathcal{C}'_B$ with a better asymptotic complexity than* `UDec` *as soon as the complexity of* `UDec` *is equal or greater than* $\mathsf{Lift}(\mathcal{C}'_B)$.

Note that the classical unique decoding algorithms over a finite fields $\mathbb{F}$ have a complexity of $O(\mathsf{M}_{\mathbb{F}}(n)\log n)$ so that the theorem holds when `UDec` is the algorithm of [Gao02] or of [Jus76].

Corollary 169. *Suppose that B is the Galois ring $\mathrm{GR}(p^r, s)$. Then there exists a unique decoding algorithm for $RS_A(v, x, k)$ with an asymptotic complexity of $\tilde{O}(rnks\log p)$ bit-operations.*

Proof. This is a direct consequence of Proposition 164, Lemma 165 and [Jus76]. □

Remark 170. We show that the gain is more significant when the arithmetic of the underlying rings and fields is not done with asymptotically fast algorithms which is the case for practical applications. By [CAY08, Table 1, page 5] the complexity of the unique decoding black box algorithm is $O(n^2)$ arithmetic operations over the alphabet when the latter is a finite field of characteristic 2.

- If $B = \mathrm{GR}(p^r, s)$ then Algorithm 16 performs at most $O(rn^2s^2\log^2 p)$ bit-operations.
- If $B = \mathbb{F}_{p^s}[[t]]/(t^r)$ then Algorithm 16 performs at most $O(rn^2s^2)$ arithmetic operations over $\mathbb{F}_p$.

Note that if B is a finite field of cardinality p^{rs} the cost of unique decoding is $O(n^2r^2s^2\log^2 p)$ bit-operations. The unique decoding becomes cheaper when the alphabet is a ring and we have a similar theorem as Theorem 168 for Galois rings.

Example 171. Let now $\mathcal{C}$ be the RS code over $\mathbb{Z}/11^3\mathbb{Z}$ with support $(1, 2, 3, 4, 5, 6, 7)$ of dimension 3. Thus $\mathcal{C}$ has parameters $[7, 3, 5]_{\mathbb{Z}/11^3\mathbb{Z}}$ by Corollary 151. Therefore its unique decoding radius is 2. Let $y = (133, 158, 163, 181, 201, 344, 247)$ be a received word. Executing algorithm 16 we get the following:

1. $i = 0$ and $y_0 = y \mod 11 = (1, 4, 9, 5, 3, 3, 5) \in \mathbb{F}_{11}$.
   The black box algorithm returns
   - $c_0 \mod p = (1, 4, 9, 5, 3, 3, 5) \in \mathcal{C}/11\mathcal{C} \subseteq \mathbb{F}_{11}^7$ and
   - $e_0 \mod p = (0, 0, 0, 0, 0, 0, 0) \in \mathbb{F}_{11}^7$,

   which can be lifted to
   - $c_0 = (1, 4, 9, 16, 25, 36, 49) \in \mathcal{C} \subseteq (\mathbb{Z}/11^3\mathbb{Z})^7$ and

- $e_0 = (0,0,0,0,0,0,0) \in (\mathbb{Z}/11^3\mathbb{Z})^7$.

2. $i = 1$ and $y_1 = (y_0 - c_0 - e_0)/11 \mod 11 = (1,3,3,4,5,6,7)$.
   The black box algorithm returns
   - $c_1 \mod p = (1,2,3,4,5,6,7) \in \mathcal{C}/11\mathcal{C}$ and
   - $e_1 \mod p = (0,1,0,0,0,0,0) \in \mathbb{F}_{11}^7$,

   which can be lifted to
   - $c_1 = (1,2,2,4,5,6,7) \in \mathcal{C}$ and
   - $e_1 = (0,1,0,0,0,0,0) \in (\mathbb{Z}/11^3\mathbb{Z})^7$.

3. $i = 2$ and $y_2 = (y_1 - c_1 - e_1)/11 \mod 11 = (1,1,1,1,1,2,1)$.
   The black box algorithm returns
   - $c_2 \mod p = (1,1,1,1,1,1,1) \in \mathcal{C}/11\mathcal{C}$ and
   - $e_2 \mod p = (0,0,0,0,0,1,0) \in \mathbb{F}_{11}^7$,

   which can be lifted to
   - $c_2 = (1,1,1,1,1,1,1) \in \mathcal{C}$ and
   - $e_2 = (0,1,0,0,0,1,0) \in (\mathbb{Z}/11^3\mathbb{Z})^7$.

4. We then build the codeword from its homogeneous components $y_0 + 11y_1 + 11^2y_2 \mod 11^3 = (133,147,163,181,201,223,247)$.

In this example the error is $e = e_0 + 11e_1 + 11^2e_2 \mod 11^3 = (0,11,0,0,0,11^2,0)$.

3.4.2 The Welch-Berlekamp algorithm

Before giving the Welch-Berlekamp decoding algorithm, we need to define what the *evaluation* of a bivariate polynomial over A is. Let $Q = \sum Q_{i,j}X^iY^j \in A[X,Y]$ be such a polynomial. We define the evaluation of Q at $(a,b) \in A^2$ to be

$$Q(a,b) = \sum Q_{i,j}b^ja^i \in A.$$

Be careful of the order of a and b. This choice will be explained in the proof of Lemma 172. Let $f \in A[X]$, we define the evaluation of Q at f to be

$$Q(X, f(X)) = \sum Q_{i,j}(f(X))^jX^i \in A[X].$$

As in the univariate case, the evaluation maps defined above are not ring homomorphisms in general.

Lemma 172. *Let $g \in A[X]$, $Q \in A[X,Y]$ of degree at most 1 in Y and $a \in A$. Then*

$$(Q(X, g(X)))(a) = Q(a, g(a)).$$

Proof. We write

$$\begin{aligned} Q(X,Y) &= Q_0(X) + Q_1(X)Y \\ &= Q_0(X) + \left(\sum_i Q_{1i}X^i\right) Y. \end{aligned}$$

The proof is an easy calculation:

$$\begin{aligned} (Q(X,g(X)))(a) &= \left(Q_0(X) + \sum_i Q_{1i}g(X)X^i\right)(a) \\ &= Q_0(a) + \sum_i Q_{1i}g(a)a^i \\ &= Q(a,g(a)) \text{ by definition.} \end{aligned}$$

□

We now adapt the Welch-Berlekamp algorithm [BW86] to noncommutative GRS. By Corollary 151, we have $\left\lfloor \frac{d-1}{2} \right\rfloor = \left\lfloor \frac{n-k}{2} \right\rfloor$. We let $\tau = \left\lfloor \frac{n-k}{2} \right\rfloor$.

Algorithm 17 Welch-Berlekamp

Input: a received vector y of A^n with at most τ errors.
Output: the unique codeword within distance τ of y.

1: $z = (z_1, \ldots, z_n) \leftarrow (v_1^{-1}y_1, \ldots, v_n^{-1}y_n)$.
2: Find $Q = Q_0(X) + Q_1(X)Y \in (A[X])[Y]$ of degree 1 such that

1. $Q(x_i, z_i) = 0$ for all $1 \leq i \leq n$,
2. $\deg Q_0 \leq n - \tau - 1$,
3. $\deg Q_1 \leq n - \tau - 1 - (k-1)$.
4. The leading coefficient of Q_1 is 1_A.

3: $f \leftarrow$ the unique root of Q in $A[X]_{<k}$.
4: **return** $(v_1 f(x_1), \ldots, v_n f(x_n))$.

In order to prove the correctness of the Welch-Berlekamp algorithm, we start with the following lemmas.

Lemma 173. *Let $y = (y_1, \ldots, y_n) \in A^n$ be such that $\tau \leq \left\lfloor \frac{n-k}{2} \right\rfloor$. Then there exists a nonzero bivariate polynomial $Q = Q_0 + Q_1 Y \in A[X,Y]$ satisfying*

1. *$Q(x_i, z_i) = 0$ for $i = 1, \ldots, n$.*
2. *$\deg Q_0 \leq n - \tau - 1$.*
3. *$\deg Q_1 \leq n - \tau - 1 - (k-1)$.*

4. The leading coefficient of Q_1 is 1_A.

Proof. We can write $y = c + e$ uniquely with $c = (v_1 f(x_1), \ldots, v_n f(x_n)) \in \mathrm{GRS}(n, k)$ for a polynomial $f \in A[X]_{<k}$ and $e \in A^n$, $w(e) \leq \tau$. Let $E = \mathrm{Supp}(e)$ and $Q_1(X) = \prod_{i \in E}(X - x_i)$. Then take $Q_0(X) = -f(X)Q_1(X)$. Then $Q(X, Y) = Q_0(X) + Q_1(X)Y$ satisfies the conditions of the lemma. □

Lemma 174. *Let $Q \in A[X, Y]$ be a bivariate polynomial satisfying the four conditions of Lemma 173 and $f \in A[X]_{<k}$ be such that $d(z, f(x)) \leq \tau$. Then $Q(X, f(X)) = 0$.*

Proof. The polynomial $Q(X, f(X))$ has degree at most $n-\tau-1$. By Lemma 172 we have $(Q(X, f(X)))(x_i) = Q(x_i, f(x_i)) = Q(x_i, z_i) = 0$ for at least $n-\tau$ values of $i \in \{1, \ldots, n\}$. And by Corollary 136 we must have $Q(X, f(X)) = 0$. □

The correctness of the algorithm is a direct consequence of Lemma 173 and 174.

Proposition 175. *Algorithm 17 works correctly as expected and can correct up to $\left\lfloor \frac{n-k}{2} \right\rfloor$ errors.*

Example 176. Let $\mathcal{C}$ be the RS code over $M_2(\mathbb{F}_7)$ with support

$$\left(\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}; \begin{pmatrix} 2 & 0 \\ 0 & 2 \end{pmatrix}; \begin{pmatrix} 3 & 0 \\ 0 & 3 \end{pmatrix}; \begin{pmatrix} 4 & 0 \\ 0 & 4 \end{pmatrix}; \begin{pmatrix} 5 & 0 \\ 0 & 5 \end{pmatrix} \right)$$

of dimension 3. Thus $\mathcal{C}$ is a $[5, 3, 2]_{M_2(\mathbb{F}_7)}$ linear code by Corollary 151. Therefore its unique decoding radius is 2. Let

$$y = \left(\begin{pmatrix} 5 & 3 \\ 2 & 2 \end{pmatrix}; \begin{pmatrix} 2 & 2 \\ 4 & 1 \end{pmatrix}; \begin{pmatrix} 4 & 0 \\ 0 & 4 \end{pmatrix}; \begin{pmatrix} 2 & 3 \\ 3 & 3 \end{pmatrix}; \begin{pmatrix} 5 & 5 \\ 0 & 6 \end{pmatrix} \right)$$

be a received word. Executing algorithm 17 we get the following:

1. By Lemma 173, Q is found using linear algebra with the affine systems of equations

$$\begin{pmatrix} 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 5 & 3 & 5 & 3 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 2 & 2 & 2 & 2 \\ 1 & 0 & 2 & 0 & 4 & 0 & 1 & 0 & 2 & 2 & 4 & 4 \\ 0 & 1 & 0 & 2 & 0 & 4 & 0 & 1 & 4 & 1 & 1 & 2 \\ 1 & 0 & 3 & 0 & 2 & 0 & 6 & 0 & 4 & 0 & 5 & 0 \\ 0 & 1 & 0 & 3 & 0 & 2 & 0 & 6 & 0 & 4 & 0 & 5 \\ 1 & 0 & 4 & 0 & 2 & 0 & 1 & 0 & 2 & 3 & 1 & 5 \\ 0 & 1 & 0 & 4 & 0 & 2 & 0 & 1 & 3 & 3 & 5 & 5 \\ 1 & 0 & 5 & 0 & 4 & 0 & 6 & 0 & 5 & 5 & 4 & 4 \\ 0 & 1 & 0 & 5 & 0 & 4 & 0 & 6 & 0 & 6 & 0 & 2 \end{pmatrix} \times \begin{pmatrix} a_0 & b_0 \\ u_0 & v_0 \\ a_1 & b_1 \\ u_1 & v_1 \\ a_2 & b_2 \\ u_2 & v_2 \\ a_3 & b_0 \\ u_3 & v_3 \\ a_4 & b_4 \\ u_4 & v_4 \\ 1 & 0 \\ 0 & 1 \end{pmatrix} = 0.$$

From this we find a solution and therefore polynomials

$$\begin{aligned} Q_0(X) &= \begin{pmatrix} a_0 & b_0 \\ u_0 & v_0 \end{pmatrix} + \begin{pmatrix} a_1 & b_1 \\ u_1 & v_1 \end{pmatrix} X + \begin{pmatrix} a_2 & b_2 \\ u_2 & v_2 \end{pmatrix} X^2 \begin{pmatrix} a_3 & b_3 \\ u_3 & v_3 \end{pmatrix} X^3 \\ &= \begin{pmatrix} 2 & 1 \\ 2 & 6 \end{pmatrix} + \begin{pmatrix} 0 & 5 \\ 0 & 3 \end{pmatrix} X + \begin{pmatrix} 2 & 4 \\ 0 & 5 \end{pmatrix} X^2 + \begin{pmatrix} 6 & 3 \\ 2 & 4 \end{pmatrix} X^3. \end{aligned}$$

and

$$\begin{aligned} Q_1(X) &= \begin{pmatrix} a_4 & b_4 \\ u_4 & v_4 \end{pmatrix} + \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} X \\ &= \begin{pmatrix} 4 & 0 \\ 0 & 4 \end{pmatrix} + \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} X. \end{aligned}$$

2. $Q(X,Y) = Q_0(X) + Q_1(X)Y$ has only one root in $M_2(\mathbb{F}_7)[X]$ by Lemma 134. It is computed with the classical Euclidean division algorithm. Thus we get the following root of Q

$$\begin{pmatrix} 3 & 5 \\ 3 & 2 \end{pmatrix} + \begin{pmatrix} 1 & 1 \\ 1 & 4 \end{pmatrix} X + \begin{pmatrix} 1 & 4 \\ 5 & 3 \end{pmatrix} X^2.$$

And then retrieve the corresponding codeword

$$c = \left(\begin{pmatrix} 5 & 3 \\ 2 & 2 \end{pmatrix}; \begin{pmatrix} 2 & 2 \\ 4 & 1 \end{pmatrix}; \begin{pmatrix} 1 & 2 \\ 2 & 6 \end{pmatrix}; \begin{pmatrix} 2 & 2 \\ 2 & 3 \end{pmatrix}; \begin{pmatrix} 5 & 5 \\ 0 & 6 \end{pmatrix} \right).$$

We can modify Algorithm 17 so that it also returns the error. In this example the error is

$$e = \left(\begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix}; \begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix}; \begin{pmatrix} 3 & 5 \\ 5 & 5 \end{pmatrix}; \begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix}; \begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix} \right).$$

We now give an example of a Reed-Solomon code defined over $M_2(\mathbb{Z}/5^2\mathbb{Z})$. We use Algorithm 16 together with Algorithm 17 to decode a word.

Example 177. Let $A = M_2(\mathbb{Z}/5^2\mathbb{Z})$. The ideal generated by $p = \begin{pmatrix} 5 & 0 \\ 0 & 5 \end{pmatrix}$ is two sided and p satisfies the condition $(*)$ of Subsection 3.4.1. Therefore we can apply Algorithm 16 to the Reed-Solomon code whose support is

$$\left(\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}; \begin{pmatrix} 2 & 0 \\ 0 & 2 \end{pmatrix}; \begin{pmatrix} 3 & 0 \\ 0 & 3 \end{pmatrix}; \begin{pmatrix} 4 & 0 \\ 0 & 4 \end{pmatrix} \right)$$

and of dimension 2. Let

$$y = \left(\begin{pmatrix} 21 & 14 \\ 14 & 22 \end{pmatrix}; \begin{pmatrix} 20 & 8 \\ 15 & 7 \end{pmatrix}; \begin{pmatrix} 5 & 17 \\ 5 & 1 \end{pmatrix}; \begin{pmatrix} 22 & 6 \\ 13 & 3 \end{pmatrix} \right)$$

be a received word. Executing Algorithm 16 we get

1. $i = 0$ and

$$y_0 \mod p = y \mod p = \left(\begin{pmatrix} 1 & 4 \\ 4 & 2 \end{pmatrix}; \begin{pmatrix} 0 & 3 \\ 0 & 2 \end{pmatrix}; \begin{pmatrix} 0 & 2 \\ 0 & 1 \end{pmatrix}; \begin{pmatrix} 2 & 1 \\ 3 & 3 \end{pmatrix} \right).$$

Algorithm 17 with input y_0 returns

$$c_0 \mod p = \left(\begin{pmatrix} 1 & 4 \\ 4 & 2 \end{pmatrix}; \begin{pmatrix} 3 & 3 \\ 2 & 4 \end{pmatrix}; \begin{pmatrix} 0 & 2 \\ 0 & 1 \end{pmatrix}; \begin{pmatrix} 2 & 1 \\ 3 & 3 \end{pmatrix} \right)$$

and the error

$$e_0 \mod p = \left(\begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix}; \begin{pmatrix} 2 & 0 \\ 3 & 3 \end{pmatrix}; \begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix}; \begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix} \right)$$

which can be lifted to the codeword

$$c_0 = \left(\begin{pmatrix} 6 & 4 \\ 4 & 2 \end{pmatrix}; \begin{pmatrix} 8 & 8 \\ 7 & 4 \end{pmatrix}; \begin{pmatrix} 10 & 12 \\ 10 & 6 \end{pmatrix}; \begin{pmatrix} 12 & 16 \\ 13 & 8 \end{pmatrix} \right).$$

We then compute

$$y_1 = (y_0 - c_0 - e_0)/p = \left(\begin{pmatrix} 3 & 2 \\ 2 & 4 \end{pmatrix}; \begin{pmatrix} 2 & 0 \\ 1 & 0 \end{pmatrix}; \begin{pmatrix} 4 & 1 \\ 4 & 4 \end{pmatrix}; \begin{pmatrix} 2 & 3 \\ 0 & 4 \end{pmatrix} \right).$$

2. $i = 1$ and

$$y_1 \mod p = \left(\begin{pmatrix} 3 & 2 \\ 2 & 4 \end{pmatrix}; \begin{pmatrix} 2 & 0 \\ 1 & 0 \end{pmatrix}; \begin{pmatrix} 4 & 1 \\ 4 & 4 \end{pmatrix}; \begin{pmatrix} 2 & 3 \\ 0 & 4 \end{pmatrix} \right).$$

Algorithm 17 with input $y_1 \mod p$ returns

$$c_0 \mod p = \left(\begin{pmatrix} 3 & 2 \\ 2 & 4 \end{pmatrix}; \begin{pmatrix} 1 & 4 \\ 3 & 4 \end{pmatrix}; \begin{pmatrix} 4 & 1 \\ 4 & 4 \end{pmatrix}; \begin{pmatrix} 2 & 3 \\ 0 & 4 \end{pmatrix} \right)$$

and the error

$$e_0 \mod p = \left(\begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix}; \begin{pmatrix} 1 & 1 \\ 3 & 1 \end{pmatrix}; \begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix}; \begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix} \right)$$

which can be lifted to the codeword

$$c_0 = \left(\begin{pmatrix} 3 & 2 \\ 2 & 4 \end{pmatrix}; \begin{pmatrix} 6 & 4 \\ 3 & 4 \end{pmatrix}; \begin{pmatrix} 9 & 6 \\ 4 & 4 \end{pmatrix}; \begin{pmatrix} 12 & 8 \\ 5 & 4 \end{pmatrix} \right).$$

3. We then return the codeword

$$c = c_0 + c_1 p \mod p^2 = \left(\begin{pmatrix} 21 & 14 \\ 14 & 22 \end{pmatrix}; \begin{pmatrix} 13 & 3 \\ 22 & 24 \end{pmatrix}; \begin{pmatrix} 5 & 17 \\ 5 & 1 \end{pmatrix}; \begin{pmatrix} 22 & 6 \\ 13 & 3 \end{pmatrix} \right).$$

In this example the error is

$$e = \left(\begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix}; \begin{pmatrix} 7 & 5 \\ 18 & 8 \end{pmatrix}; \begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix}; \begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix} \right).$$

3.5 List decoding of generalized Reed-Solomon codes

3.5.1 List-decoding over certain valuation rings

In this subsection, as in Subsection 3.4.1 we let A be a ring satisfying $(*)$, S be the set formed by the powers of p, $\mathcal{C} = \mathrm{GRS}_A(v, x, k)$ be a GRS code and G a generator matrix of $\mathcal{C}$ and $\Lambda = A/(p)$ be the residual ring. We precise our black box list decoding algorithm.

Algorithm 18 Black box list decoding algorithm

Input: a received vector y of Λ^n with at most τ errors.
Output: a subset S of $\Lambda^k \times \Lambda^n$ such that $(m, e) \in S \Leftrightarrow mG + e = y$ and $w(e) \leq \tau$.

The list decoding algorithm we propose is recursive and the following algorithm is its recursive step.

Algorithm 19 List decoding from valuation i up to valuation r.

Input: two nonnegative integers $i \leq r$, a received vector y of A^n with at most τ errors. A **black box** list decoding algorithm as specified by Algorithm 18 for the code $\mathcal{C}/p\mathcal{C}$ for decoding up to τ errors.
Output: The set $U \stackrel{def}{=} \{c \in \mathcal{C} : d(c \mod p^{r-i}, y \mod p^{r-i}) \leq \tau\}$.

1: **if** $i = r$ **then**
2: **return** $\{0\}$.
3: **end if**
4: Call the **black box** algorithm with input $(y \mod p)$ to obtain a subset $S \subseteq \Lambda^k \times \Lambda^n$.
5: **for** each $(m_0, e_0) \in S$ **do**
6: $c_0 \leftarrow m_0 G$.
7: Call recursively Algorithm 19 with arguments $i + 1$, r and $y_{c_0} = (y - c_0 - e_0)/p$ to get the set S_{c_0} of all the codewords in the ball centered in y_{c_0} of radius τ.
8: **end for**
9: **return** $\{c_0 + c_1 p : c_0 \in S \text{ and } c_1 \in S_{c_0} \text{ and } d(c_0 + pc_1 \mod p^{r-i}, y \mod p^{r-i}) \leq \tau\}$.

Proposition 178. *Algorithm 19 is correct and can decode up to τ errors.*

Proof. The proof is done by descending induction on i. If $i = r - 1$ the proposition holds.

Now let $i < r-1$, $c \in U$, $e = y-c$. There exists $(m_0, e_0) \in S$ such that $c_0 = m_0 G = c \mod p$. Then by Lemma 162 $(c_0 - c)/p \in \mathcal{C}$. Moreover we have $\mathrm{Supp}(e_0) \subseteq \mathrm{Supp}(e)$ and $w(e) \leq \tau$. Therefore $w((e_0 - e)/p) \leq \tau$. We have $y_{c_0} = (y - (c_0 + e_0))/p = (c - c_0)/p + (e_0 - e)/p$ and by the inductive hypothesis there exists $c_1 \in S_{c_0}$ such that $(c - c_0)/p = c_1 \mod p^{r-(i+1)}$. □

The complexity of Algorithm 19 will be studied in detail in Subsection 3.5.3 when the ring A is finite. We now give an algorithm for list decoding a GRS code over $B = A/(p^r)$ for a positive integer r.

Algorithm 20 List decoding over a valuation ring.

Input: a positive integer τ, a received vector y of B^n with at most τ errors and a **black box** unique decoding algorithm for $\mathcal{C}/p\mathcal{C}$.

Output: the list of codewords within distance τ of y.

1: $z \leftarrow$ a representative of y in A^n.
2: Call Algorithm 19 with parameters 0, r, and z and obtain the set T.
3: **return** $\{c \mod p^r : c \in T\}$.

Proposition 179. *Algorithm 20 works correctly as expected.*

Proof. This is a direct consequence of Proposition 178 □

Example 180. In this example we work with the $\mathrm{RS}_{\mathbb{Z}/7^2\mathbb{Z}}(6,2)$ code whose support is $(1,2,3,4,5,6)$. The unique decoding radius is 2 while the list decoding algorithm radius is 3. Suppose we received the word $y = (8,15,22,11,12,13) \in (\mathbb{Z}/7^2\mathbb{Z})^6$. We skip steps 1, 2 and 4 of Algorithm 20 and identify the elements of $\mathbb{Z}_7$ up to precision 2 with the elements of $\mathbb{Z}/7^2\mathbb{Z}$ for the clarity of the example. The execution of Algorithm 20 is as follows:

- We enter Algorithm 19 with $y = (8,15,22,11,12,13)$.
- At step 4, the call to the black box algorithm with $y \mod 7 = (1,1,1,4,5,6)$ returns two codewords and their corresponding errors (step 5):
  1. The codeword $(1,1,1,1,1,1)$ which can be lifted to $(1,1,1,1,1,1)$ and the error $(0,0,0,3,4,5)$.
  2. The codeword $(1,2,3,4,5,6)$ which can be lifted to $(1,2,3,4,5,6)$ and the error $(0,1,2,0,0,0)$.
- We have a list of two candidates, for each one we do a recursive call of Algorithm 19.
- For item 1:
  - We enter *recursively* Algorithm 19 with $[y - (1,1,1,1,1,1) - (0,0,0,3,4,5)]/7 = (1,2,3,1,1,1)$.
  - At step 4 the call to the black box algorithm with $(1,2,3,1,1,1)$ returns the two codewords $(1,1,1,1,1,1,1)$ and $(1,2,3,4,5,6)$ which can be lifted to $(1,1,1,1,1,1,1)$ and $(1,2,3,4,5,6)$ (step 5).
  - At step 9, we return $(1,1,1,1,1,1,1)$ and $(1,2,3,4,5,6)$.
- For item 2:

  - We enter *recursively* Algorithm 19 with $[y - (1,2,3,4,5,6) - (0,1,2,0,0,0)]/7 = (1,1,1,1,1,1)$.
  - At step 4 the call to the black box algorithm with $(1,1,2,1,1,1)$ returns the codeword $(1,1,1,1,1,1,1)$ which can be lifted to $(1,1,1,1,1,1,1)$.
  - At step 9, we return $(1,1,1,1,1,1,1)$.

- Due to the condition of step 9 of Algorithm 19 we return only the two codewords
  - $(1,1,1,1,1,1) + 7 \times (1,2,3,4,5,6) \mod 7^2 = (8,15,22,29,36,43)$ and
  - $(1,2,3,4,5,6) + 7 \times (1,1,1,1,1,1) \mod 7^2 = (8,9,10,11,12,13)$.

  The codeword $(1,1,1,1,1,1) + 7 \times (1,1,1,1,1,1) = (8,8,8,8,8,8)$ is not returned at step 9 because $d(y,(8,8,8,8,8,8)) = 5 > J = 3$.

3.5.2 The Guruswami-Sudan algorithm

We now extend the Guruswami-Sudan [GS98] algorithm to noncommutative GRS codes. We assume in this section that $\{x_1,\ldots,x_n\} \subseteq Z(A)$. Almost nothing has to be changed from the original algorithm. In this subsection we do the following assumption on A: every linear system with coefficients in A with more unknowns than equations has a nonzero solution (with coefficients also in A). This is the case for example when A is finite.

Lemma 181. *Let A be any finite ring, $n < m$ two positive integers and $M \in M_{n\times m}(A)$. Then there exists a nonzero $x \in A^m$ such that $Mx = 0$.*

Proof. The matrix M defines a left A-linear map $A^m \to A^n$ and therefore a morphism of abelian groups. The kernel of this morphism has cardinality at least $|A^m/A^n| = |A|^{m-n} > 0$. □

Lemma 182. *Let $y \in A^n$ be a received word with at most τ errors with $\tau < J$. Then there exists a nonzero bivariate polynomial $Q \in A[X,Y]$ satisfying the three conditions of step 3 of Algorithm 21.*

Proof. As usual we consider the coefficients of Q to be unknowns satisfying the equations $Q(x_i,z_i) = 0$ and $[Q(X+x_i,Y+z_i)]_{s'} = 0$ for $i = 1,\ldots,n$ and $s' = 0,\ldots,s-1$ where $[Q]_{s'}$ denotes the homogeneous component of degree s' of Q.

First note that the value of s in step 2 together with $\tau < J$ imply

$$\left[(n-\tau)^2 - (k-1)n\right] s^2 - (k-1)s - 1 > 0,$$

which in turn implies

$$s^2(n-\tau)^2 - 1 > n(k-1)(s^2+s),$$

and then gives

$$\left(\frac{s(n-\tau)-1}{k-1}\right)\left(\frac{s(n-\tau)+1}{2}\right) > n\binom{s+1}{2}.$$

Algorithm 21 Guruswami-Sudan

Input: a positive integer $\tau < J$ and a received vector y of A^n with at most τ errors.
Output: all the $f \in A[X]_{<k}$ such that $d(y, f(x)) \leq \tau$.

1: $s \leftarrow \left\lfloor \frac{(k-1)n+\sqrt{(k-1)^2n^2+4((n-\tau)^2-(k-1)n)}}{2((n-\tau)^2-(k-1)n)} \right\rfloor + 1$.
2: $L \leftarrow \left\lceil \frac{s(n-\tau)-1}{k-1} \right\rceil - 1$.
3: $z = (z_1, \ldots, z_n) \leftarrow (v_1^{-1}y_0, \ldots, v_n^{-1}y_n)$.
4: Find $Q = \sum_{i=0}^{L} Q_i(X)Y^i \in (A[X])[Y]$ of degree at most L such that

1. $Q(x_i, z_i) = 0$ for all $1 \leq i \leq n$.
2. $Q(X + x_i, Y + z_i)$ has valuation at least s.
3. $\deg Q_i \leq s(n-\tau) - 1 - i(k-1)$ for all $0 \leq i \leq L$.

5: $\mathcal{Z} \leftarrow$ Roots of Q in $A[X]_{<k}$ such that $d(z, f(x)) \leq \tau$.
6: **return** $\{(v_1 f(x_1), \ldots, v_n f(x_n) : f \in \mathcal{Z}\}$

Counting the coefficients of Q we get

$$(L+1)\left(s(n-\tau) - (k-1)\frac{L}{2}\right)$$

unknowns which is greater or equal than

$$\left(\frac{s(n-\tau)-1}{k-1}\right)\left(\frac{s(n-\tau)+1}{2}\right).$$

On the other hand conditions 1 and 2 of step 3 of Algorithm 21 give $n\binom{s+1}{2}$ equations. And we have a nonzero solution by the hypothesis made on A. □

Lemma 183. *Let $g \in A[X]$, $Q \in A[X,Y]$ and $z \in Z(A)$. Then*

$$(Q(X, g(X)))(a) = Q(a, g(a)).$$

Proof. According to the definition we took for evaluating polynomials in Subsection 3.4.2, we have:

$$\begin{aligned}(Q(X, g(X)))(a) &= \sum (Q_{ij}(g(X))^j X^i)(a) \\ &= \sum Q_{ij}(g(a))^j a^i \text{ because } a \in Z(A) \\ &= Q(a, g(a)) \text{ by definition.}\end{aligned}$$

□

Remark 184. Note that for the Guruswami-Sudan algorithm we could have defined the evaluation of bivariate polynomials in the "usual way" that is, for $f \in A[X]$, $Q \in A[X,Y]$ and $a, b \in A$,

$$Q(a,b) = \sum_{i,j} Q_{ij} a^i b^j$$

and

$$Q(X, f(X)) = \sum_{i,j} Q_{ij} X^i (f(X))^j.$$

As the evaluation is done at points from the center of A both definitions for evaluation give the exact same result.

Lemma 185. *Let $Q \in A[X,Y]$ verifying the three conditions of step 1 of Algorithm 21. Let $f \in A[X]_{<k}$ such that $f(x_i) = z_i$ for a fixed $i \in \{1, \ldots, n\}$. Then $(X - x_i)^s$ divides $Q(X, f(X))$.*

Proof. By assumption we have

$$Q(X + x_i, Y + z_i) = \sum_{\lambda \geq s} \sum_{j+l=\lambda} Q_{jl} X^j Y^l$$

with $Q_{jl} \in A$ and where $s' \geq s$ is the valuation of Q. By Remark 135, there exists a polynomial $g(X) \in A[X]$ such that $f(X) - z_i = g(X)(X - x_i)$. As $x_i \in Z(A)$ we have

$$\begin{aligned} Q(X, f(X)) &= Q((X - x_i) + x_i, (f(X) - z_i) + z_i) \\ &= \sum_{\lambda \geq s'} \sum_{j+l=\lambda} Q_{jl} (g(X)(X - x_i))^l (X - x_i)^j \\ &= \sum_{\lambda \geq s'} \sum_{j+l=\lambda} Q_{jl} g(X)^l (X - x_i)^\lambda \\ &= (X - x_i)^{s'} h(X) \end{aligned}$$

where $h(X) \in A[X]$. □

Lemma 186. *Let $Q \in A[X,Y]$ be a bivariate polynomial satisfying the three conditions of step 4 of Algorithm 21 and let $f \in A[X]_{<k}$ be such that $d(y, f(x)) \leq \tau$. Then $Q(X, f(X)) = 0$.*

Proof. Let $f \in A[X]_{<k}$ be a polynomial such that $d(y, f(x)) \leq \tau$. Then $Q(X, f(X))$ is a polynomial of degree at most $s(n - \tau) - 1$. We have $f(x_i) = z_i$ for at least $n - \tau$ values of i. By Lemma 183, $(Q(X, f(X)))(x_i) = Q(x_i, f(x_i)) = 0$ for at least $n - \tau$ values of i.

Denote by E the set $\{i \in \{1, \ldots, n\} : Q(x_i, f(x_i))) = 0\}$ and by $P_r(X)$ the polynomial $\prod_{i \in E} (X - x_i)^r$. We prove by induction on $r \leq s$ that $P_r(X)$ divides $Q(X, f(X))$. For $r = 1$ it is a consequence of Remark 135 and the assumption we made on the support x of the code. By induction there exists $R(X) \in A[X]$ such that $Q(X, f(X)) = R(X) P_r(X)$. Let $i_0 \in E$, by Lemma 185 we also have $Q(X, f(X)) = S(X)(X - x_{i_0})^{r+1}$. By Lemma 134

we have $S(X)(X - x_{i_0}) = R(X)\prod_{i\in E, i\neq i_0}(X - x_i)^r$, whence $R(x_{i_0}) = 0$. This is true for all $i_0 \in E$ and by Remark 135 and the property of the support x we deduce that $P_{r+1}(X)$ divides $Q(X, f(X))$.

It follows that $Q(X, f(X))$ is divisible by a monic polynomial of degree $s(n - \tau)$ which implies $Q(X, f(X)) = 0$. □

Proposition 187. *Algorithm 21 works correctly as specified and can correct up to* $\lceil J \rceil - 1$ *errors.*

Proof. This is direct consequence of Lemma 182 and Lemma 186. □

3.5.3 Complexities for list decoding algorithms

In order to study the complexity of Algorithm 20, we need a result about the number of codewords that can be returned by Algorithm 21. The results of [NH00, Section 5] remain valid in our context. In other words, they do not depend on the algebraic structure of the alphabet. We recall them in the following proposition for the sake of completeness. We assume throughout this subsection that all errors with weight at most τ occur with the same probability regardless of the weight of the transmitted codeword.

Proposition 188. *We let* $\mathcal{C} = \mathrm{GRS}_A(n, k)$, $c \in \mathcal{C}$, $w = w(c)$;

$$N(c, a, i) = 0 \text{ if } w > a + i,$$

and

$$N(c,a,i) = \sum_{j=0}^{\min\left(\left\lfloor \frac{i-(w-a)}{2} \right\rfloor, n-w\right)} \left[\binom{w}{w-a+j} \binom{n-w}{j} (|A|-1)^j \binom{a-j}{i-(w-a)-2j} (|A|-2)^{i-(w-a)-2j} \right]$$

else. We now let

$$M(\tau) = \sum_{u=d}^{\min(2\tau, n)} \sum_{a=u-\tau}^{\tau} \sum_{i=u-a}^{\tau} N(c, a, i).$$

Then the probability that Algorithm 21 returns more than one codeword is at most

$$\frac{M(\tau)}{\sum_{i=0}^{\tau} \left[\binom{n}{i} (|A|-1)^i \right]}.$$

Proof. The proof is identical to [NH00, Proposition 12, page 9]. □

In Tables 3.1 and 3.2 we give examples of this upper bound as it is difficult to get a simple asymptotic equivalent. These calculations have been made for finite fields in [NH00, Section 5] and for Galois rings in [Arm05b, Section 5]. As shown in the tables the probability is very small.

r (nilpotency index of $p = 3$)	Upper bound
1	0.001310
2	1.386×10^{-6}
3	1.850×10^{-9}
4	2.530×10^{-12}
5	3.469×10^{-15}
6	4.759×10^{-18}
7	6.528×10^{-21}
8	8.954×10^{-24}
9	1.228×10^{-26}
10	1.685×10^{-29}

Figure 3.1: Table for $\mathrm{RS}_{\mathrm{GR}(3^r,2)}[8, 4, 5]$ and a codeword of weight 6.

ℓ (matrix size)	Upper bound
1	0.001310
2	2.530×10^{-12}
3	1.228×10^{-26}
4	1.123×10^{-46}
5	1.930×10^{-72}
6	6.247×10^{-104}
7	3.804×10^{-141}
8	4.358×10^{-184}
9	9.396×10^{-233}
10	3.812×10^{-287}

Figure 3.2: Table for $\mathrm{RS}_{M_\ell(\mathbb{F}_9)}[8, 4, 5]$ and a codeword of weight 6.

Remark 189. As pointed out in the introduction of this subsection the upper bound on the probability given in Proposition 188 is independent of the algebraic structure of the alphabet. Therefore there is no gain in taking the Galois ring or a matrix ring (over a finite field or a Galois ring) instead of the finite field of same cardinality. In fact the advantage resides in the asymptotic complexity of the decoding algorithms given in Subsection 3.4.1 and 3.5.1.

We now let, as in Subsection 3.5.2, $J = n - \sqrt{(k-1)n}$ be the generic Johnson bound. We recall the following proposition:

Proposition 190. *Let $y \in A^n$. Then there exist at most $n(|A|-1)$ codewords within distance J from y.*

Proof. See [Gur04, Corollary 3.3 page 36]. □

We can now state the proposition about the complexity of Algorithm 20.

Proposition 191. *With the same notations as in Subsection 3.5.1, let $\mathsf{LDec}(\mathcal{C})$ be the complexity in terms of the number of bit-operations of a list decoding algorithm for $\mathrm{GRS}_{A/(p)}(\mathcal{C})$ and ρ be the probability that the latter algorithm returns more than one codeword. We suppose that $\mathsf{LDec}(\mathcal{C})$* does not *depend on ρ. Then Algorithm 20*

- *performs at most*

$$\frac{[n(|A|-1)]^r - 1}{n(|A|-1)-1}(\mathsf{LDec}(\mathcal{C}) + \mathsf{Lift}(\mathcal{C})) = (n|A|)^{r-1}(\mathsf{LDec}(\mathcal{C}) + \mathsf{Lift}(\mathcal{C})).$$

  bit-operations.

- *performs an* expected *number of at most*

$$\frac{[(1-\rho) + \rho(n(|A|-1))]^r - 1}{[(1-\rho) + \rho(n(|A|-1))] - 1}(\mathsf{LDec}(\mathcal{C}) + \mathsf{Lift}(\mathcal{C})).$$

  bit-operations.

Proof. It is a direct consequence of Proposition 188 and 190. □

Remark 192. Taking the notations of Proposition 191 Recall that $\mathsf{LDec}(\mathcal{C})$ does not depend on ρ. *Heuristically* it is interesting to see that as $\rho \to 0$ we have

$$\frac{[(1-\rho) + \rho(n(|A|-1))]^r - 1}{[(1-\rho) + \rho(n(|A|-1))] - 1}(\mathsf{LDec}(\mathcal{C}) + \mathsf{Lift}(\mathcal{C})) \\ \to (r + \rho o(1))\,(\mathsf{LDec}(\mathcal{C}) + \mathsf{Lift}(\mathcal{C})) = r(\mathsf{LDec}(\mathcal{C}) + \mathsf{Lift}(\mathcal{C}))$$

Therefore the complexity of Algorithm 20 is *heuristically* polynomial in r, n and $|A|$ whenever $\mathsf{LDec}(\mathcal{C})$ and $\mathsf{Lift}(\mathcal{C})$ are polynomial in r, n and $|A|$. This *heuristic* analysis is reasonable according to Tables 3.1 and 3.2. Therefore we can notice, as in Proposition 166, that if we denote by B the Galois ring $\mathrm{GR}(p^r, s)$ then the asymptotic complexity given above is better than the complexity of the corresponding decoding algorithm over the finite field of size p^{rs}.

Corollary 193. *Suppose that A is the Galois ring $\mathrm{GR}(p^r, s)$. Then there exists a list decoding algorithm for $GRS(v, x, k)$ with an asymptotic complexity of*

$$\tilde{O}\left(\frac{n^r(\rho^{rs}-1)^r-1}{n(\rho^{rs}-1)-1} n^7 k^5 s \log p\right) = \tilde{O}(n^{r+6}\rho^{r(r-1)s} k^5 s \log p)$$

arithmetic operations in $\mathbb{F}_p$, or heuristically *an expected number of $\tilde{O}(rn^7k^5 s \log p)$ bit-operations which can decode up to the generic Johnson bound.*

Proof. This is a direct consequence of Proposition 191, Remark 192, Lemma 165 and [Gur04, Lemma 6.13, page 111]. □

Note that the algorithm presented in [Gur04, Algorithm *Poly-Reconstruct*, page 102] applied to a RS code over $\mathbb{F}_{p^{rs}}$ performs at most $\tilde{O}(n^7k^5rs)$ arithmetic operations in $\mathbb{F}_p$. We have an *heuristic* result for list decoding similar to Theorem 130. Let $\mathfrak{D}$ be the list decoding algorithm of [Ale05, AZ08, GS98, Köt96, KV03, RR98, Sud97b].

Heuristic Result 194. *Given a finite field A, a Galois ring B such that $|A| = |B|$, a RS code $\mathcal{C}_A$ over A of parameters $[n, k, n-k+1]_A$ and a list decoding algorithm* `LDec` *from the list $\mathfrak{D}$ for $\mathcal{C}_A$. Suppose that there exists a RS code $\mathcal{C}_B$ over B of parameters $[n, k, n-k+1]_B$. Then there exists a list decoding algorithm for $\mathcal{C}_B$ with a better asymptotic complexity than* `LDec`.

3.6 Conclusion

In this paper we showed that, with strong constraints on their supports, GRS codes can be considered over non commutative rings. But this generalization does not lead to better codes than GRS codes over commutative rings in terms of the parameters.

We also proposed two new decoding algorithms with a low complexity for GRS codes over Galois rings and rings of matrices over a Galois ring. Using these algorithms we showed that given a prime power q and a unique (resp. list) decoding algorithm for a GRS code over $\mathbb{F}_q$ there exists a unique (resp. list) decoding algorithm for a GRS code with same parameters (provided that the GRS code exists with our conditions on its support) over $\mathbb{Z}/q\mathbb{Z}$ with a better asymptotic complexity.

Acknowledgment

The authors would like to thank Daniel Augot for his precious advices and Alain Couvreur and Grégoire Lecerf for their careful readings of this article.

Chapter 4

A Lifting Decoding Scheme and its Application to Interleaved Linear Codes

This chapter constitutes an accepted paper at ISIT (International Symposium on Information Theory) 2012.

Abstract—In this paper we design a decoding algorithm based on a lifting decoding scheme. This leads to a unique decoding algorithm with complexity quasi linear in all the parameters for Reed-Solomon codes over Galois rings and a list decoding algorithm. We show that, using erasures in our algorithms, allows one to decode more errors than half the minimum distance with a high probability. Finally we apply these techniques to interleaved linear codes over a finite field and obtain a decoding algorithm that can recover more errors than half the minimum distance.
Keywords—Algorithm design and analysis, Decoding, Error correction, Reed-Solomon codes, Interleaved codes.

4.1 Introduction

Reed-Solomon (RS) codes form an important and well-studied family of codes. They can be efficiently decoded. See for example [Gao02, Jus76]. They are widely used in practice [WB99]. Sudan's 1997 breakthrough on list decoding of RS codes [Sud97b], further improved by Guruswami and Sudan in [GS98], showed that RS codes are list decodable up to the Johnson bound in polynomial time.

4.1.1 Our contributions

Let B be a quotient ring of a discrete valuation ring A with uniformizing parameter π. We design a decoding scheme that can be adapted to a wide range of linear codes over B. Let $\mathcal{C}$ be a code over B, then given a black box decoding algorithm `BlackBoxDec`

for $\mathcal{C}/\pi\mathcal{C}$, we can construct a decoding algorithm for $\mathcal{C}$ generalizing [GV98, algorithm of Section 3]. The constructed decoding algorithm has the property to correct all error patterns that can be corrected by `BlackBoxDec`. We study in detail the complexities in the case of Reed-Solomon codes over Galois rings and truncated power series rings over a finite field.

We improve the construction given in [GV98, algorithm of Section 3] and in [BZ01, Byr01] by integrating an idea used by Marc Armand in [AdT05, Arm05a]. We use erasures at suitable places within our decoding algorithm to improve its decoding radius. This improvement allows one to decode more error patterns than `BlackBoxDec` with a high probability. We study and give complexities when RS codes are involved. In fact, we decode exactly the same error patterns as in Armand's papers [AdT05, Arm05a] but with a lower complexity thanks to the decoding scheme of [GV98].

Finally we show that, given any linear code $\mathcal{C}'$ over $\mathbb{F}_q$, we can view interleaved codes with respect to $\mathcal{C}'$ as codes over $\mathbb{F}_q[[t]]/(t^r)$. This allows one to apply the previous techniques to interleaved codes to obtain a decoding algorithm that can decode more errors than half the minimum distance of $\mathcal{C}'$ with a high probability over small alphabets (small finite fields). Our approach is different from [BKY03], which treats *a priori* only the case of interleaved RS codes while our algorithm is able to decode (further than half the minimum distance) any interleaved linear code as soon as a decoding algorithm for the underlying code is available. Therefore we can consider codes over small alphabet like $\mathbb{F}_2$. A lot of families of codes are subfield-subcodes of alternant codes. Thus a lot of interleaved codes can be decoded with the approach of [BKY03] but at a higher cost than our approach which does not need to consider alternant codes.

4.1.2 Related work

Our approach for a lifting decoding scheme has first been studied in [GV98], then in [BZ01, Byr01] RS codes over a commutative finite ring have been studied by M. Armand in [Arm04, Arm05b, Arm05a, AdT05]. The decoding of interleaved codes has been studied in [BKY03, CS03, GGR11].

4.2 Prerequisites

4.2.1 Complexity model

The "soft-Oh" notation $f(n) \in \tilde{O}(g(n))$ means that $f(n) \in g(n)\log^{O(1)}(3+g(n))$. It is well known [Für07] that the time needed to multiply two integers of bit-size at most n in *binary representation* is $\tilde{O}(n)$. The cost of multiplying two polynomials of degree at most n over a ring A is $\tilde{O}(n)$ in terms of the number of arithmetic operations in A. Thus the bit-cost of multiplying two elements of the finite field $\mathbb{F}_{p^n}$ is $\tilde{O}(n\log p)$.

4.2.2 Error correcting codes

In this section we let A be any commutative ring with identity and n be a positive integer. Let C be a subset of A^n. We call C an *error correcting code over* A or simply a *code over* A. If C is a submodule of A^n we say that C is a *linear* code over A. The integer n is called the *blocklength* of C. If C is a linear code and C is free of rank k, then we say that C has *parameters* $[n, k]_A$.

Definition 195. Let $u = (u_1, \ldots, u_n) \in A^n$. We call the integer

$$w(u) := |\{i \in \{1, \ldots, n\} : u_i \neq 0\}|$$

the *Hamming weight* (or simply *weight*) of u. Let v be another vector of A^n. The integer $w(u - v)$ is called the *Hamming distance* (or simply *distance*) between u and v and is denoted by $d(u, v)$.

The integer $d = \min_{u,v \in C \text{ and } u \neq v} d(u, v)$ is called the *minimum distance of* C. Note that when C is a linear code we have $d = \min_{u \in \mathcal{C} \setminus \{0\}} w(u)$, we then say that C has parameters $[n, k, d]_A$ if C is free of rank k.

Definition 196. Suppose that C is free of rank k. A matrix whose rows form a basis of C is called a *generator matrix* of C.

The generator matrix is used to encode a *message* $m \in A^k$. A generator matrix induces a one-to-one correspondence between messages and codewords, the map $m \mapsto mG$ is a A-linear embedding $A^k \to A^n$. Under this map, we will identify messages and codewords.

Let $\mathfrak{m}$ be a maximal ideal of A. The vector space $C/\mathfrak{m}C$, if not zero, is a linear code with parameters $[n, \leq k, \leq d]_{A/\mathfrak{m}}$ with generator matrix G'. The matrices G and G' have the same number of columns but can have a different number of rows. However G' can be deduced from G, first compute $G'' = G \mod \mathfrak{m}$, then remove from G'' appropriate rows to obtain a basis of $C/\mathfrak{m}C$.

Definition 197. Borrowing the terminology of [GV98, Section 3], if G and G' have the same number of rows and columns and that $G \mod \mathfrak{m} = G'$ then $\mathcal{C}$ is called a *splitting* code.

We will consider codes over a special kind of rings which we define now.

Definition 198. Let A be a ring. If A is a local principal ideal domain, we call A a *discrete valuation ring* (DVR). Any element $\pi \in A$ such that (π) is the maximal ideal of A is called a *uniformizing parameter* of A.

4.2.3 Reed-Solomon codes over rings

Reed-Solomon codes over rings are defined in a slightly different way to their field counterparts. We let $A[X]_{<k}$ denote the submodule of $A[X]$ consisting of all the polynomials of degree at most $k - 1$ of $A[X]$.

Definition 199. Let $x_1, \ldots, x_n$ be elements of A such that $x_i - x_j \in A^\times$ fir $i \neq j$ (where $A^\times$ is the group of units of A). The submodule of A^n generated by the vectors $(f(x_1), \ldots, f(x_n)) \in A^n$ where $f \in A[X]_{<k}$ is called a *Reed-Solomon* code over A. The n-tuple $(x_1, \ldots, x_n)$ is called the *support* of the RS code.

Proposition 200. *Let $\mathcal{C}$ be a RS code over A. Then $\mathcal{C}$ has parameters $[n, k, d = n - k + 1]_A$.*

Proposition 201. *Let $\mathcal{C}$ be a RS code with parameters $[n, k, d = n - k + 1]_A$ over a discrete valuation ring A with uniformizing parameter π. Then $\mathcal{C}/\pi^r\mathcal{C}$ is a RS code with parameters $[n, k, d]_{A/(\pi^r)}$ over $A/(\pi^r)$. Moreover of $(x_1, \ldots, x_n)$ is the support of $\mathcal{C}$ then $(x_1 \mod \pi^r, \ldots, x_n \mod \pi^r)$ is the support of $\mathcal{C}/\pi^r\mathcal{C}$.*

4.3 Improved π-adic lifting.

In this section we let A be a discrete valuation ring with uniformizing parameter π and by $\kappa = A/(\pi)$ the residue field of A. We also let $\mathcal{C}$ be a free splitting linear code over A of parameters $[n, k, d]_A$ and with generator matrix G. We let $\mathcal{C}'$ denote the linear code $\mathcal{C}/\pi\mathcal{C}$ and G' a generator matrix of $\mathcal{C}'$ such that $G' = G \mod \pi$.

Algorithm 22 `BlackBoxDec`

Input: A positive integer $\tau \leq n$ and a received vector y of κ^n (with zero or more erasures).

Output: A nonempty set $U \subseteq \kappa^k \times \kappa^n$ satisfying

$$(m, e) \in U \Rightarrow y = mG' + e \text{ and } w(e) \leq \tau \tag{4.1}$$

or $\emptyset$ (meaning FAILURE).

Note that `BlackBoxDec` can return one or more codewords in particular it can be a list decoding algorithm; but we do not require that it return *all* codewords within distance τ of y.

Proposition 202. *Suppose that `BlackBoxDec` returns* all *the codewords from $\mathcal{C}'$ within distance τ of $y \in \kappa^n$. Then Algorithm 23 can decode up to τ errors up to precision r.*

Proof. The proof is done by descending induction on i. For $i = r$ and $i = r - 1$ the proposition holds.

Now let $i < r-1$ and $(m, e) \in \kappa^k \times \kappa^n$. Let $c = mG$ be such that $w(e \mod \pi^{r-i}) \leq \tau$ and $y = c+e$. There exists $(m_0, e_0) \in S$ such that $c_0 = m_0G = c \mod \pi$, $e = e_0 \mod \pi$ and $\mathrm{Supp}(e_0) \subseteq \mathrm{Supp}(e)$. If we count erasures as errors, we have $w(e) \leq \tau$ and therefore $w(\pi^{-1}(e_0 - e)) \leq \tau$. On the other hand we have $mG = m_0G \mod \pi$ and $mG' = m_0G'$ in $\mathcal{C}'$ whence $m = m_0 \mod \pi$. Therefore $\pi^{-1}(mG - m_0G) = (\pi^{-1}(m - m_0))G \in \mathcal{C}$.

We deduce from the above that

$$y_1 = \pi^{-1}(y - (c_0 + e_0)) = \pi^{-1}(c - c_0) + \pi^{-1}(e_0 - e).$$

Algorithm 23 Decoding from valuation i up to valuation r.

Input: A positive integer $\tau \leq n$, two nonnegative integers $i \leq r$, a received vector y of A^n (with zero or more erasures) and a black box decoding algorithm `BlackBoxDec` for $\mathcal{C}(\pi)$.

Output: A nonempty set $U \subseteq \kappa^k \times \kappa^n$ satisfying

$$(m, e) \in U \Rightarrow y = mG + e \mod \pi^{r-i} \text{ and } w(e) \leq \tau \tag{4.2}$$

or $\emptyset$ (meaning FAILURE).

1: **if** $i = r$ **then**
2: **return** $\{(0, 0)\}$.
3: **end if**
4: Call to `BlackBoxDec` with input τ and $(y \mod \pi)$ to obtain the set S.
5: **if** `BlackBoxDec` returns $\emptyset$ (FAILURE) **then**
6: **return** $\emptyset$ (FAILURE).
7: **end if**
8: $U \leftarrow \emptyset$.
9: **for** each $(m_0, e_0) \in S$ **do**
10: $y_1 \leftarrow \pi^{-1}(y - m_0 G - e_0)$.
11: Put erasures in y_1 at the locations indicated by $\mathrm{Supp}(e_0)$.
12: Call recursively Algorithm 23 with input τ, $i+1$, r, y_1 and `BlackBoxDec` to obtain the set T.
13: **for** each $(m_1, e_1) \in T$ **do**
14: **if** $|\mathrm{Supp}(e_0) \cup \mathrm{Supp}(e_1)| \leq \tau$ **then**
15: $U \leftarrow U \cup \{(m_0 + \pi m_1, e_0 + \pi e_1)\}$.
16: **end if**
17: **end for**
18: **end for**
19: **return** U.

Algorithm 24 Decoding up to precision r.

Input: A positive integer $\tau \leq n$, a positive integer r, a received vector y of A^n (with zero or more erasures) and a black box decoding algorithm `BlackBoxDec` for $\mathcal{C}(\pi)$.

Output: A nonempty set $U \subseteq \kappa^k \times \kappa^n$ satisfying

$$(m, e) \in U \Rightarrow y = mG' + e \mod \pi^{r} \text{ and } w(e) \leq \tau \tag{4.3}$$

or $\emptyset$ (meaning FAILURE).

1: **return** the set returned by the call to Algorithm 23 with input τ, 0, r, y and `BlackBoxDec`.

By the inductive hypothesis, we can find $(m_1, e_1) \in T$ such that $\pi^{-1}(c - c_0) = m_1 G \mod \pi^{r-(i+1)}$ and $\pi^{-1}(e - e_0) = e_1 \mod \pi^{r-(i+1)}$. □

We now have the straightforward proposition which gives the complexity of Algorithm 24 in terms of bit operations.

Proposition 203. *Suppose that the number of codewords returned by* `BlackBoxDec` *is at most $L > 1$. Denote by $\mathsf{Lift}(\mathcal{C})$ the complexity of lifting a codeword of $\mathcal{C}'$ into a codeword of $\mathcal{C}$ up to precision r in terms of the number of bit operations. Denote by $\mathsf{UDec}(\mathcal{C})$ the complexity of algorithm* `BlackBoxDec` *in terms of the number of bit operations. Then Algorithm 24 performs at most*

$$\frac{L^r - 1}{L - 1} \left(\mathsf{Lift}(\mathcal{C}) + \mathsf{UDec}(\mathcal{C})\right) = O(L^{r-1}) \left(\mathsf{Lift}(\mathcal{C}) + \mathsf{UDec}(\mathcal{C})\right)$$

bit operations. If $L \leq 1$ then Algorithm 24 performs at most $r \left(\mathsf{Lift}(\mathcal{C}) + \mathsf{UDec}(\mathcal{C})\right)$ bit operations.

Algorithm 25 Decoding algorithm for $\mathcal{C}/\pi^r\mathcal{C}$.

Input: A positive integer $\tau \leq n$, a received vector y of $(A/(p^r))^n$ (with zero or more erasures) and a black box decoding algorithm `BlackBoxDec` for $\mathcal{C}(\pi)$.

Output: A nonempty set $U \subseteq \kappa^k \times \kappa^n$ satisfying

$$(m, e) \in U \Rightarrow y = mG' + e \text{ and } w(e) \leq \tau \tag{4.4}$$

or $\emptyset$ (meaning FAILURE).

1: Lift $y \in (A/(p^r))^n$ into $y' \in A^n$.
2: $S \leftarrow$ the set returned by the call to Algorithm 23 with input τ, 0, r, y' and `BlackBoxDec`.
3: **return** $\{c \mod \pi^r : c \in S\}$.

The interesting part of Algorithm 23 (and hence of all other algorithms) resides in the `BlackBoxDec` argument. We have shown that if `BlackBoxDec` is a classical decoding algorithm then Algorithm 24 becomes a decoding algorithm with the same decoding radius as `BlackBoxDec`.

From now we suppose that $\kappa = A/(\pi)$ is a finite field. Every element of $B = A/(\pi^r)$ can be uniquely written as $u\pi^s$, where $u \in B^\times$ and $0 \leq s \leq r - 1$.

RS codes are free splitting codes over B by Proposition 201 so we can apply Algorithm 25 to RS codes. Complexities of decoding with Algorithm 25 are given by the following proposition which is a direct consequence of Proposition 203.

Example-proposition 204. *Suppose that $\mathcal{C}$ is a RS code over B. If $B = \mathrm{GR}(p^r, s)$ (the unique Galois extension over $\mathbb{Z}/p\mathbb{Z}$ of degree s) then*

- *if* `BlackBoxDec` *is the unique decoding algorithm of [Jus76] (that can decode up to $\tau = \left\lfloor \frac{d-1}{2} \right\rfloor$ errors) then Algorithm 25 can decode up to τ errors in $\tilde{O}(rnks \log p)$ bit operations,*

- *if* `BlackBoxDec` *is the Guruswami-Sudan list decoding algorithm of [Gur04, Corollary 3.3, page 36] (that can decode up to* $J = \left\lceil n - \sqrt{(k-1)n} \right\rceil - 1$ *errors) then Algorithm 25 can list decode up to* J *errors in* $\tilde{O}\left([n(|\kappa|-1)]^{r-1} n^7 k^5 s \log p\right)$ *bit operations.*

If $B = \kappa[[t]]/(t^r)$ *then*

- *if* `BlackBoxDec` *is the unique decoding algorithm of [Jus76] (that can decode up to* $\tau = \left\lfloor \frac{d-1}{2} \right\rfloor$ *errors) then Algorithm 25 can decode up to* τ *errors in* $\tilde{O}(rnk)$ *arithmetic operations over* κ*.*
- *if* `BlackBoxDec` *is the Guruswami-Sudan list decoding algorithm of [Gur04, Corollary 3.3, page 36] (that can decode up to* $J = \left\lceil n - \sqrt{(k-1)n} \right\rceil - 1$ *errors) then Algorithm 25 can list decode up to* J *errors in* $\tilde{O}\left([n(|\kappa|-1)]^{r-1} n^7 k^5\right)$ *arithmetic operations over* κ*.*

We show that if we choose a decoding algorithm able to handle errors and erasures for `BlackBoxDec` then we can decode, with a non negligible probability, further than half the minimum distance and further than the Johnson bound.

Definition 205. Following the terminology of [Laz65, Subsection 2.1, page 404] we say that an element of B has *filtration* s if it is written $u\pi^s$ where $u \in B^\times$.

We let q be the cardinality of κ. Then the cardinality of B is q^r while the cardinality of $\frac{A/(\pi^s)}{A/(\pi^{s+1})}$ is q.

Algorithm 26 `BlackBoxErasuresDec`

Input: A received vector y of κ^n with ϵ erasures and at most $\tau(\epsilon)$ errors.
Output: All the codewords within distance $\tau(\epsilon) + \epsilon$ of y or $\emptyset$ (FAILURE).

Proposition 206. *Let* $\mathcal{C}$ *be a splitting code over* B *with parameters* $[n,k]_B$*. Suppose that* ϵ *erasures occurred and that* `BlackBoxErasuresDec` *is provided as the* `BlackBoxDec` *argument to Algorithm 25. The number of error vectors of weight* w *that can be corrected by Algorithm 25 is at least*

$$N(\epsilon, B, w) = \binom{n}{\epsilon} q^{r\epsilon} \binom{n-\epsilon}{w} \times \sum_{(v_0,\ldots,v_{r-1}) \in V_w} \left[\prod_{i=0}^{r-1} \binom{w - v_0 - \cdots - v_{i-1}}{v_i} (q-1)^{v_i} q^{v_0+\cdots+v_{i-1}} \right] \quad (4.5)$$

where

$$V_w = \{(v_0,\ldots,v_{r-1}) \in \mathbb{N}^r : v_0 + \cdots + v_{r-1} = w \text{ and } 0 \le v_0 \le \tau(\epsilon) \text{ and } 0 \le v_{i-1} \le \tau(\epsilon + v_0 + \cdots + v_{i-2}) \text{ for } i = 2,\ldots,r-1\},$$

hence the fraction of corrigible error patterns is at least

$$P(\epsilon, B, w) = \frac{\sum_{i=0}^{w} N(\epsilon, B, w)}{\sum_{i=0}^{w} \binom{n}{i} (q^r - 1)^i} \tag{4.6}$$

Proof. Let $e \in B^n$ be an error vector. We let $v_i(e)$ for $i = 1, \ldots, r-1$ denote the number of coordinates of e of filtration i. The number of error vectors $e \in B^n$ such that $(v_0(e), \ldots, v_{r-1}(e)) \in V_w$ is given by formula 4.5. Let c be a codeword of $\mathcal{C}$ and $y = c + e$ with $v_i = v_i(e)$ for $i = 0, \ldots, r-1$ and $(v_0, \ldots, v_{r-1}) \in V_w$. The rest of the proof is similar to the proof of Proposition 202. □

Proposition 207. *Let $\mathcal{C}$ be a splitting code over B with parameters $[n, k, d]_B$. Then there exists a decoding algorithm such that $\tau(\epsilon) = \left\lfloor \frac{d-\epsilon-1}{2} \right\rfloor$.*

Proof. This is a consequence of [Rot06, Theorem 1.7, page 16]. □

Proposition 208. *Let $\mathcal{C}$ be a Reed-Solomon code over B with parameters $[n, k, d = n - k + 1]_B$ then there exists*

- *a unique decoding algorithm which can correct errors and erasures with $\tau(\epsilon) = \left\lfloor \frac{n-\epsilon-k}{2} \right\rfloor$,*
- *a list decoding algorithm which can correct errors and erasures with $\tau(\epsilon) = \left\lceil (n-\epsilon) - \sqrt{(k-1)(n-\epsilon)} \right\rceil - 1$ and*
- *a unique decoding algorithm which can correct up to w errors and ϵ erasures with $w \leq n - \epsilon - k$ and which does succeed for a fraction of at least $P(\epsilon, B, w)$ error patterns.*

In addition the costs of Algorithm 25 are the same as the ones given in Proposition 204.

Proof. For the first item, see for example [Gao02, Section 4, page 7 and 8] while for the second item see [GS98, Theorem 16, page 1762]. The third item is a consequence of the first item and Proposition 206. □

4.4 Application to interleaved linear codes.

In this section we let A be the power series ring over the finite field $\mathbb{F}_q$ namely we let $A = \mathbb{F}_q[[t]]$, $\pi = t$ and $B = \mathbb{F}_q[[t]]/(t^r)$. We recall the construction of interleaved codes and show that all interleaved codes over $\mathbb{F}_q$ are exactly codes over B. We let $\mathcal{C}'$ be a linear code over $\mathbb{F}_q$ with parameters $[n, k, d]_{\mathbb{F}_q}$ and with generator matrix G'.

Let r messages $m_0, \ldots, m_{r-1} \in \mathbb{F}_q^k$ and their encodings $c_0 = m_0 G', \ldots, c_{r-1} = m_{r-1} G'$. For $i = 0, \ldots, r-1$ and $j = 1, \ldots, n$ define c_{ij} to be the j-th coordinate of c_i and $s_j = (c_{0,j}, \ldots, c_{r-1,j})$.

$$\begin{array}{|c|c|c|c|} \hline c_{0,1} & c_{0,2} & \cdots & c_{0,n} \\ \hline c_{1,1} & c_{1,2} & \cdots & c_{1,n} \\ \hline \vdots & \vdots & \vdots & \vdots \\ \hline c_{r-1,1} & c_{r-1,2} & \cdots & c_{r-1,n} \\ \hline \end{array} \begin{array}{l} \rightarrow \ c_0 \\ \rightarrow \ c_1 \\ \\ \rightarrow \ c_{r-1} \end{array}$$
$$\begin{array}{cccc} \downarrow & \downarrow & & \downarrow \\ s_1 & s_2 & & s_n \end{array}$$

The vectors transmitted over the channel are not $c_1, \ldots, c_{r-1} \in \mathbb{F}_q^n$ but $s_1, \ldots, s_n \in \mathbb{F}_q^r$. We will make an abuse of notation and call such an encoding scheme a *interleaved code with respect to $\mathcal{C}'$ and of degree r*. Usually the vector s_j (for $j = 1, \ldots, n$) is seen as an element of $\mathbb{F}_{q^r}$, but we can associate the element $\sum_{i=0}^{r-1} c_{i,j} t^i \in B$ to s_j. In this context, if $y = (y_1, \ldots, y_n) \in (\mathbb{F}_q^r)^n$, the weight of y is the nonnegative integer $|\{i \in \{1, \ldots, n\} : y_i \neq 0\}|$ and if y corresponds to the received word then the weight of the error is $|\{i \in \{1, \ldots, n\} : y_i \neq s_i\}|$.

Proposition 209. *The words transmitted over the channel using interleaved linear codes are precisely the transmitted words using linear codes over truncated power series.*

Proof. Let $G = G'$ be the generator of the linear code $\mathcal{C}$ over B with parameters $[n, k, \leq d]_B$, then $\mathcal{C}/t\mathcal{C} = \mathcal{C}'$. We have $c_i = m_i G'$ for $i = 0, \ldots, r-1$. As a consequence we have

$$\left(\sum_{i=0}^{r-1} m_i t^i\right) G = \sum_{i=0}^{r-1} (m_i G)\, t^i = \sum_{i=0}^{r-1} c_i t^i$$
$$= \left(\sum_{i=0}^{r-1} c_{i,1} t^i, \sum_{i=0}^{r-1} c_{i,2} t^i, \ldots, \sum_{i=0}^{r-1} c_{i,n} t^i\right) = (s_1, s_2, \ldots, s_n)\,.$$

This shows that the transmitted words using interleaved linear codes correspond exactly to codewords of $\mathcal{C}$. Moreover the weight of $(s_1, \ldots, s_n)$ as defined above is the same as the Hamming weight of $\left(\sum_{i=0}^{r-1} c_{i,1} t^i, \sum_{i=0}^{r-1} c_{i,2} t^i, \ldots, \sum_{i=0}^{r-1} c_{i,n} t^i\right) \in \mathcal{C}$. □

Theorem 210. *Given a linear code $\mathcal{C}'$ over $\mathbb{F}_q$ with parameters $[n, k, d]_{\mathbb{F}_q}$ and a unique decoding algorithm* `BlackBoxErasuresDec` *from errors and erasures that can correct ϵ erasures and $\tau(\epsilon)$ errors in $\operatorname{dec}(\mathcal{C}')$ arithmetic operations over $\mathbb{F}_q$, there exists a unique decoding algorithm for interleaved codes with respect to $\mathcal{C}'$ and of degree r from errors and erasures that can correct ϵ erasures and $\tau(\epsilon)$ errors with at most $r \operatorname{dec}(\mathcal{C}')$ arithmetic operations over $\mathbb{F}_q$. Moreover it can correct at least a fraction of $P(\epsilon, B, w)$ error patterns of Hamming weight at most $w > \tau(\epsilon)$ over B where P is defined by 4.6, also with at most $r \operatorname{dec}(\mathcal{C}')$ arithmetic operations over $\mathbb{F}_q$.*

Proof. As $G = G'$ there is no need to lift a codeword from $\mathcal{C}'$ into $\mathcal{C}$ and the given complexities are a consequence of Proposition 203. The existence of both algorithm is ensured by Proposition 209 and Proposition 206. □

	2	3	4	5	6
7	1.0	1.0	1.0	1.0	1.0
8	0.96	0.98	0.99	0.99	0.99
9	0.81	0.94	0.96	0.97	0.98
10	0.49	0.80	0.88	0.91	0.91
11	0.0073	0.53	0.70	0.75	0.78
12	0.00012	0.14	0.38	0.48	0.53

Figure 4.1: Fraction of corrigible error patterns for a Goppa code of parameters $[256, 200, 15]_{\mathbb{F}_2}$.

	3	4	5	6
22	1.00000	1.00000	1.00000	1.00000
23	0.999997	0.999999	0.999999	0.999999
25	0.999844	0.999963	0.999981	0.999987
27	0.998099	0.999469	0.999715	0.999789
28	0.995114	0.998531	0.999185	0.999391
29	0.989079	0.996477	0.997984	0.998470
30	0.978112	0.992458	0.995554	0.996581

Figure 4.2: Fraction of corrigible error patterns for an Extended BCH code with parameters $[256, 100, 46]_{\mathbb{F}_2}$.

In Tables 4.1 and 4.2, the first row gives the degrees of interleaving and the first column shows the number of errors up to which we want to decode. The second row corresponds to half the minimum distance and, as expected, all of the probabilities are 1.0. We can see that the fraction of corrigible error patterns increases with the degree of interleaving and that codes with a high minimal distance are good candidates for interleaving.

4.5 Conclusion

In this paper we designed a decoding algorithm based on a lifting decoding scheme. It allowed us to obtain a unique decoding algorithm for RS codes over Galois rings with a low complexity. We also applied this scheme to get a list decoding algorithm for RS codes over Galois rings. We then show that using erasures at appropriate positions in the proposed algorithms allows us to decode more errors than half the minimum distance. Finally we applied these techniques to decode interleaved linear codes over a finite field and get a decoding algorithm that can decode more errors than half the minimum distance.

Acknowledgment

The author would like to thank Daniel Augot for his precious advice and readings of this article and Grégoire Lecerf for his careful readings of this article. The author would also like to thank the reviewers who helped improve this article.

Part III

Related work on error correcting codes

Context

In this part I present other results obtained during my PhD thesis. They concern quasi cyclic codes whose definition is given above and number fields codes which are introduced later in this introduction.

Quasi cyclic codes

Let $n = m\ell$ be three positive integers.

Definition 211. Let $\mathcal{C} \subseteq \mathbb{F}_q^n$ be a code. We say that $\mathcal{C}$ is *cyclic* if

$$(c_1, \ldots, c_{n-1}, c_n) \in \mathcal{C} \Rightarrow (c_n, c_1, \ldots, c_{n-1}) \in \mathcal{C}$$

and we say that $\mathcal{C}$ is *ℓ-quasi-cyclic* if

$$(c_1, \ldots, c_n) \in \mathcal{C} \Rightarrow (c_{n-\ell+1}, \ldots, c_n, c_1, \ldots, c_{n-\ell}) \in \mathcal{C}.$$

It is well known [MS86a, Theorem 1, page 190] that there is a one-to-one correspondence between cyclic codes and ideals of the ring $\mathbb{F}_q[X]/(X^n - 1)$. The goal of our paper is to extend this correspondence to quasi cyclic codes.

Number fields codes

Number fields codes form a subfamily of *Chinese remaindering theorem*-codes (CRT codes) which were first studied in [GSS00] then in [Gur04, Chapter 7, page 147–175].

Definition 212. Let A be any commutative ring with identity and $I_1, \ldots, I_n$ be coprime ideals and let $E \subseteq A$. Then the CRT code denoted by $\mathrm{CRT}((I_1, \ldots, I_n), E)$ is the set

$$\{(x \mod I_1, \ldots, x \mod I_n) \,|\, x \in E\}.$$

When $A = \mathbb{Z}$, $I_1 = (p_1), \ldots, I_n = (p_n)$ where $p_1, \ldots, p_n$ are primes and

$$E = \left\{ x \in \mathbb{Z} | 0 \leq x < \prod_{i=0}^{k} p_i \right\}$$

where $k < n$ we obtain a well studied class of codes [Man76, GRS99, Bon00, GSS00, Gur04]. This subfamily of CRT codes is *badly* called "CRT codes" in the literature. We will call them "CRT codes over $\mathbb{Z}$".

There is a Johnson bound (Definition 15) for CRT codes as soon as $|A/I_i| < +\infty$ for all $i \in \{1, \ldots, n\}$. The Johnson bound of Definition 15 works when the error correcting code $\mathcal{C}$ is a subset of A^n. This is not the case for CRT codes as one have

$$\mathcal{C} \subseteq \prod_{i=1}^{n} |A/I_i| \text{ where } |A/I_i| < +\infty \text{ and, } \textit{a priori}, |A/I_i| \neq |A/I_j|.$$

Theorem 213. *Let $A_1, \ldots, A_n$ be n finite sets such that $|A_i| = q_i$ for $i = 1, \ldots, n$ and $\mathcal{C}$ be a code of minimal distance d of $A_1 \times \cdots \times A_n$. Then*

$$n - \sqrt{n(n-d)}$$

is a Johnson bound for $\mathcal{C}$.

The proof can be found in [Gur04, Theorem 7.10, page 163]. Number fields codes are a subfamily of CRT codes we describe now.

Definition 214. Let K be a finite extension of $\mathbb{Q}$. The *Hermitian norm* of an element $x \in K$ is

$$\|x\| := \sqrt{\sum_{i=1}^{[K:\mathbb{Q}]} |\sigma_i(x)|^2}$$

where the σ_i are the embeddings $K \to \mathbb{C}$.

Definition 215. Let K be a finite extension of $\mathbb{Q}$ and $\mathcal{O}_K$ be the integral closure of $\mathbb{Z}$ in K. Let $\mathfrak{p}_1, \ldots, \mathfrak{p}_n$ be n integral prime ideals. The CRT code

$$\mathrm{CRT}\left((\mathfrak{p}_1, \ldots, \mathfrak{p}_n), \{x \in \mathcal{O}_K | \|x\| \leq B\}\right).$$

where B is a fixed integer is called a *number field* code.

As $|\mathcal{O}_K/\mathfrak{p}_i| < +\infty$ Theorem 213 can be applied and we obtain the following Johnson bound for a number field code of minimum distance d

$$n - \sqrt{n(n-d)}.$$

Number fields codes have been studied in only two papers [Len86, Gur03]. Their list decoding has been quickly considered in [CH11]. The authors used the Coppersmith theorem to claim that they can list decode number fields codes. Unfortunately they did not give any algorithm, complexity or comparison of their decoding radius with the Johnson bound. It turns out that a direct application of their theorem shows that it does not reach the Johnson bound.

Contributions

In Chapter 5 we show that ℓ-quasi-cyclic codes, like cyclic codes, are in one-to-one correspondence with left ideals of the ring $M_\ell(\mathbb{F}_q)[X]/(X^m - 1)$, give their generator matrix and a key equation. Then in Chapter 6 we give the first list decoding algorithm for number fields codes that can decode up to the Johnson bound.

Chapter 5

On Quasi-Cyclic Codes as a Generalization of Cyclic Codes

This chapter constitutes an accepted paper at FFA (Finite Fields and Their Applications) in 2012. It has been done in collaboration with Morgan Barbier and Christophe Chabot.

Abstract—In this article we see quasi-cyclic codes as block cyclic codes. We generalize some properties of cyclic codes to quasi-cyclic codes. We show a one-to-one correspondence between ℓ-quasi-cyclic codes of length $m\ell$ and left ideals of $M_\ell(\mathbb{F}_q)[X]/(X^m-1)$. Then, we generalize BCH codes and evaluation codes in this context. We study their parameters and establish a key equation. Finally, we present a new $[189, 11, 125]_{\mathbb{F}_4}$ code beating the known minimum distance for fixed length and dimension. Many codes with good parameters beating best known ones have been found from this latter.
Keywords—quasi-cyclic codes, left ideals, matrix rings, cyclic codes, evaluation codes, key equation

5.1 Introduction

5.1.1 Context

Many codes with best known minimum distances are quasi-cyclic codes or derived from them [LS03, Gra07]. This family of codes is therefore very interesting. Quasi-cyclic codes were studied and applied in the context of McEliece's cryptosystems [McE78, BCGO09] and Niederreiter's [Nie86, LDW94]. They permit to reduce the size of keys in opposition to Goppa codes. However, since the decoding of random quasi-cyclic codes is difficult, only quasi-cyclic alternant codes were proposed for the latter cryptosystems. The high structure of alternant codes is actually a weakness and two cryptanalysis were proposed in [FOPT10, UL10]. For these reasons, studying the decoding methods and the general properties of quasi-cyclic codes are interesting topics.

In [LF01, LS01], ℓ-quasi-cyclic codes of length $m\ell$ are seen as R-submodules of R^ℓ for a certain ring R. However, in [LF01], Gröbner bases are used in order to describe

polynomial generators of quasi-cyclic codes whereas in [LS01], the authors decompose quasi-cyclic codes as direct sums of shorter linear codes over various extensions of $\mathbb{F}_q$ (when $\gcd(m, q) = 1$). This last work leads to an interesting trace representation of quasi-cyclic codes. In [CCN10], the approach is more analogous to the cyclic case. The authors consider the factorization of $X^m - 1 \in M_\ell(F_q)[X]$ with reversible polynomials in order to construct ℓ-quasi-cyclic codes canceled by those polynomials and called $\Omega(P)$-codes. This leads to the construction of self-dual codes and codes beating known bounds. But the factorization of univariate polynomials over a matrix ring remains difficult. In [Cha11] the author gives an improved method for particular cases of the latter factorization problem.

In this article, we prove, analogously to the cyclic case, a one-to-one correspondence between ℓ-quasi-cyclic codes of length $m\ell$ and left ideals of $M_\ell(F_q)[X]/(X^m - 1)$. We study the properties of quasi-cyclic codes and propose to extend the definition of BCH and evaluation codes to the context of quasi-cyclic codes. Namely, we define *quasi-BCH* and *quasi-evaluation* codes. The natural notion of *folded* and *unfolded* codes is presented for simplicity and decoding purposes. Finally, we exhibit a quasi-cyclic code whose parameters are better than the previous known and 48 other codes derived from the first one.

Subsection 5.1.2 is devoted to some recalls about $\Omega(P)$-codes and definitions. Then in Section 5.2 we prove interesting properties about quasi-cyclic codes and, in particular, the correspondence between left ideals and quasi-cyclic codes. Section 5.3 deals with the definition, parameters and a decoding algorithm of quasi-BCH codes. Finally, Section 5.5 introduces quasi-evaluation codes and gives lower bounds on their parameters.

5.1.2 First definitions

In this section, we fix a positive integer n and let $\mathcal{C}$ be a code of length n over the finite field $\mathbb{F}_q$, *i.e.* a vector subspace of $\mathbb{F}_q^n$.

Definition 216 (Quasi-cyclic codes)**.** From now and until the end of this article we define $T : \mathbb{F}_q^n \to \mathbb{F}_q^n$ to be the left cyclic shift defined by:

$$T(c_1, c_2, \ldots, c_n) = (c_2, c_3, \ldots, c_1).$$

Suppose that ℓ divides n. Then we call an *ℓ-quasi-cyclic code over $\mathbb{F}_q$ of length n* a code of length n over $\mathbb{F}_q$ stable by T^ℓ. If the context is clear we will simply say *ℓ-quasi-cyclic code*.

Let ℓ be an integer, and $\alpha \in \mathbb{F}_{q^\ell}$ be such that $(1, \alpha, \ldots, \alpha^{\ell-1})$ is an $\mathbb{F}_q$-base of the vector space $\mathbb{F}_{q^\ell}$. We define the *folding* to be the $\mathbb{F}_q$-linear map

$$\begin{array}{rcl} \phi : \mathbb{F}_q^\ell & \to & \mathbb{F}_{q^\ell} = \mathbb{F}_q[\alpha] \\ (a_1, \ldots, a_\ell) & \mapsto & a_1 + a_2\alpha + \cdots + a_\ell\alpha^{\ell-1}. \end{array}$$

The unfolding is the inverse $\mathbb{F}_q$-linear map

$$\begin{array}{rcl} \phi^{-1} : \mathbb{F}_{q^\ell} & \to & \mathbb{F}_q^\ell \\ a = a_1 + a_2\alpha + \cdots + a_\ell\alpha^{\ell-1} & \mapsto & (a_1, a_2, \ldots, a_\ell). \end{array}$$

Let m be a positive integer, $f : E \to F$ be any map of sets. We denote by $f^{\times m}$ the map of sets $f^{\times m} : E^m \to F^m$ such that $f^{\times m}(x_1, \ldots, x_m) = (f(x_1), \ldots, f(x_m))$.

Definition 217 (Folded and unfolded codes)**.** Suppose that $n = m\ell$. We define the *folded code* of $\mathcal{C}$ to be $\phi^{\times m}(\mathcal{C})$. Let $\mathcal{C}'$ be a code in $\mathbb{F}_{q^\ell}^m$. We define the *unfolded code* of $\mathcal{C}'$ to be $(\phi^{-1})^{\times m}(\mathcal{C}')$.

Remark 218. Observe that a code $\mathcal{C}$ is ℓ-quasi cyclic if and only if its folded $\mathcal{C}' = \phi^{\times m}(\mathcal{C})$ is cyclic. But $\mathcal{C}'$ is not necessarily $\mathbb{F}_{q^\ell}$-linear.

5.2 Properties of quasi-cyclic codes

In the present section we generalize the results of [MS86a, Theorem 1, page 190] to quasi-cyclic codes. We fix a positive integer n and suppose that $n = m\ell$ for two positive integers m and ℓ.

5.2.1 The one-to-one correspondence

It is well-known [MS86a, Theorem 1, page 190] that there is a one-to-one correspondence between cyclic codes of length n over $\mathbb{F}_q$ and monic factors of $X^n - 1 \in \mathbb{F}_q[X]$ *i.e.* ideals of $\mathbb{F}_q[X]/(X^n - 1)$. In [CCN10,Cha11] the authors start to exhibit such a correspondence for quasi-cyclic codes. They show that there is a correspondence between a subfamily of ℓ-quasi-cyclic codes of length $m\ell$ over $\mathbb{F}_q$ and reversible factors of $X^n - 1 \in M_\ell(\mathbb{F}_q)[X]$.

The one-to-one correspondence between ℓ-quasi cyclic codes and left ideals of $M_\ell(\mathbb{F}_q)[X]/(X^m - 1)$ is a consequence of the two following lemmas. In what follows we consider principal ideal rings which are not necessarily integral domains.

Lemma 219. *Let R be a commutative principal ideal ring and M be a free left module of finite rank s over R. Then every submodule N of M can be generated by at most s elements.*

Proof. It is an easy adaptation of the proof of [Lan02, Theorem 7.1, page 146]. □

Lemma 220. *Let s be a positive integer and R be a commutative principal ideal ring. Then there is a one-to-one correspondence between the submodules of R^s and the left ideals of $M_s(R)$.*

Proof. Note that this is a particular case of the Morita equivalence for modules. See for example [Bou11, n°4, page 99]. This particular case can be proved directly. To a submodule $N \subseteq R^s$, we can build a left ideal of $M_s(R)$ whose elements have rows in N. Conversely, to a left ideal $I \subseteq M_s(R)$ we associate the submodule of R^s generated by all the rows of all the elements of I. It is straightforward to check that these maps are inverse to each other. □

Note that $M_\ell(\mathbb{F}_q)[X]/(X^m-1)$ and $M_\ell(\mathbb{F}_q[X]/(X^m-1))$ are isomorphic as rings and that $R=\mathbb{F}_q[X]/(X^m-1)$ is a commutative principal ideal ring. By Lemma 219 any submodule of R^ℓ can be generated by at most ℓ elements. Therefore by Lemma 220 any left ideal of $M_\ell(R)=M_\ell(\mathbb{F}_q)[X]/(X^m-1)$ is principal.

Theorem 4. *There is a one-to-one correspondence between ℓ-quasi-cyclic codes over $\mathbb{F}_q$ of length $m\ell$ and left ideals of $M_\ell(\mathbb{F}_q)[X]/(X^m-1)$.*

Proof. Let $g=(g_{11},\ldots,g_{1\ell},g_{21},\ldots,g_{2\ell},\ldots,g_{m1},\ldots,g_{m\ell})\in\mathbb{F}_q^{m\ell}$. We associate to g the element $\varphi(g)\in(\mathbb{F}_q[X]/(X^m-1))^\ell$ defined by

$$\begin{aligned}\varphi(g)=\big(&g_{11}+g_{21}X+\cdots+g_{m1}X^{m-1};\\ &g_{12}+g_{22}X+\cdots+g_{m2}X^{m-1};\ldots;\\ &g_{1\ell}+g_{2\ell}X+\cdots+g_{m\ell}X^{m-1}\big).\end{aligned}$$

Then φ induces a one-to-one correspondence between ℓ-quasi-cyclic codes of length $m\ell$ over $\mathbb{F}_q$ and submodules of $(\mathbb{F}_q[X]/(X^m-1))^\ell$. The theorem follows by Lemma 220. □

Let $\mathrm{pr}_{i,j}$ be the projection of the $i,i+1,\ldots,j$ coordinates:

$$\begin{array}{rcl}\mathrm{pr}_{i,j}:\mathbb{F}_q^n&\longrightarrow&\mathbb{F}_q^{j-i+1}\\(x_1,\ldots,x_n)&\longmapsto&(x_i,x_{i+1},\ldots,x_{j-1},x_j).\end{array}$$

We have the following obvious lemma:

Lemma 221. *Let $\mathcal{C}$ be an ℓ-quasi-cyclic code over $\mathbb{F}_q$ of dimension k and length $m\ell$. Then there exists an integer r such that $1\le r\le k$ and for any generator matrix G of $\mathcal{C}$ and $0\le i\le m-1$, the rank of the $i\ell+1,i\ell+2,\ldots,(i+1)\ell$ columns of G is r.*

Definition 222 (Block rank)**.** Taking the notation of Lemma 221, we call the integer r the *block rank* of $\mathcal{C}$. Note that r depends only on $\mathcal{C}$ and not on any particular generator matrix of $\mathcal{C}$.

5.2.2 The generator polynomial of an ℓ-quasi-cyclic code

In this subsection we fix an ℓ-quasi-cyclic code $\mathcal{C}$ over $\mathbb{F}_q$. If $\ell=1$, then $\mathcal{C}$ is a cyclic code of length n and a generator matrix of $\mathcal{C}$ can be given [MS86a, Theorem 1, (e), page 191] by

$$\begin{pmatrix}g(X)&&&\\&Xg(X)&&\\&&\cdots&\\&&&X^{n-\deg g}g(X)\end{pmatrix},\tag{5.1}$$

where $g(X)\in\mathbb{F}_q[X]$ is the generator polynomial of $\mathcal{C}$. The block rank of $\mathcal{C}$ is 1 and we see that we can write a generator matrix of $\mathcal{C}$ with only 1 vector and its shifts (by $T^\ell=T$). The natural generalization of this result for quasi-cyclic codes is done using the block rank.

Let r be the block rank of $\mathcal{C}$, the following algorithm computes a basis of $\mathcal{C}$ from r vectors of $\mathcal{C}$ and their shifts. We call the *first index* of a nonzero vector $x = (x_1, \ldots, x_{m\ell})$ the least integer $0 \leq i \leq m-1$ such that $(x_{i\ell+1}, \ldots, x_{(i+1)\ell}) \neq 0$ and denote it by $\mathcal{F}(x) = \mathcal{F}(x_1, \ldots, x_{m\ell})$. Let

$$\begin{array}{rcl} p : \mathbb{F}_q^{m\ell} & \longrightarrow & \mathbb{F}_q^{\ell} \\ x = (x_1, \ldots, x_{m\ell}) & \longmapsto & (x_{i\ell+1}, \ldots, x_{(i+1)\ell}), \end{array}$$

where $i = \mathcal{F}(x_1, \ldots, x_n)$ if $x \neq 0$ and $p(0) = 0$.

Algorithm 27 Basis computation with the block rank

Input: A generator matrix G of $\mathcal{C}$.
Output: A generator matrix formed by r rows from G and some of their shifts.

$G' \leftarrow$ a row echelon form of G.
Denote by $g_1, \ldots, g_k$ the rows of G'.
$M \leftarrow \max\{\mathcal{F}(g_i) : i \in \{0, \ldots, m-1\}\}$.
$B'_M \leftarrow \emptyset$.
$G_{M+1} \leftarrow \emptyset$.
for $j = M \to 0$ **do**
  $B_j \leftarrow \{g_i : i \in \{1, \ldots, k\}$ and $\mathcal{F}(g_i) = j\}$.
  for each element x of B_j **do**
    if $p(B'_j) \cup \{p(x)\}$ are independent **then**
      $B'_j \leftarrow B'_j \cup \{x\}$.
    end if
  end for
  $G_j \leftarrow G_{j+1} \cup B'_j$.
  $B'_{j-1} \leftarrow T^{\ell}(B'_j)$.
end for
return G_0.

Note that Algorithm 27 applied to a cyclic code, *i.e.* $\ell = 1$, returns exactly the matrix 5.1 and we can deduce the generator polynomial of $\mathcal{C}$ at the cost of the computation of a row echelon form of any generator matrix of $\mathcal{C}$.

Proposition 223. *Algorithm 27 works correctly as expected and returns a generator matrix G of $\mathcal{C}$ made of r linearly independent vectors of $\mathcal{C}$ and some of their shifts.*

Proof. We will prove by descending induction on j that:

1. $B'_j \supseteq T^{\ell}(B'_{j+1}) \supseteq \cdots \supseteq T^{(M-j)\ell}(B'_M)$.
2. $\#B'_j \leq r$.
3. The vectors of B'_j are linearly independent.
4. The vectors of G_j are linearly independent.

5. $\langle G_j \rangle = \langle g_i : i \in \{1, \ldots, k\} \text{ and } \mathcal{F}(g_i) \geq j \rangle$.

Let $j = M$. By step 3, we have $B_M \neq \emptyset$. Item 1 is trivially satisfied. By Lemma 221, $\#B_M \leq r$ and item 2 is satisfied. As $G_{M+1} = B'_M = \emptyset$ then $G_M = B'_M = B_M = \{g_i : i \in \{1, \ldots, k\} \text{ and } \mathcal{F}(g_i) \geq M\}$ and items 3 to 5 are satisfied.

Suppose that $j < M$ and that items 1 to 5 are satisfied for $i = j+1, \ldots, M$. First note that $B_j \neq \emptyset$. If we had $B_j = \emptyset$ then, as G' is in row echelon form, $g_1, \ldots, g_k, T^{(M-j)\ell}(g_k)$ would be linearly independent which is a contradiction.

Items 1 and 3 are satisfied by steps 7, 9 and 10 of the algorithm. By Lemma 221 and step 9, item 2 is satisfied. For all $x \in G_{j+1}$, we have $\mathcal{F}(x) \geq j + 1$, thus, by item 3, the elements of G_j are linearly independent and item 4 is satisfied. Let g be a vector of G' such that $\mathcal{F}(g) = j$, then the construction of B'_j implies that we have

$$\mathcal{F}\left(g - \sum_{u \in B'_j} \mu_u u\right) \geq j + 1$$

where $\mu_u \in \mathbb{F}_q$ for $u \in B'_j$. Then by item 5 of the inductive hypothesis, we have

$$\left(g - \sum \mu_u u\right) \in G_{j+1}.$$

Thus we have $\langle G_j \rangle = \langle g_i : i \in \{1, \ldots, k\} \text{ and } \mathcal{F}(g_i) \geq j \rangle$ and item 5 is satisfied.

As a consequence of the previous induction, G_0 is constituted of linearly independent vectors and generates $\langle g_i : i \in \{1, \ldots, k\} \text{ and } \mathcal{F}(g_i) \geq 0 \rangle = \mathcal{C}$ by item 5. By Lemma 221 we must have exactly r vectors $g \in G_0$ such that $\mathcal{F}(g) = 0$. Thus by items 1 and 2 we have

$$r = \#B'_0 = \sum_{\lambda=0}^{M} \# \left(B'_\lambda \setminus T^\ell(B'_{\lambda+1})\right)$$

which shows that G_0 is constituted of r linearly independent vectors of $\mathcal{C}$ and some of their shifts. □

Corollary 224. *There exist $g_1, \ldots, g_r$ linearly independent vectors of $\mathcal{C}$ such that $g_1, \ldots, g_r, T^\ell(g_1), \ldots, T^\ell(g_r), \ldots, T^{(m-1)\ell}(g_1), \ldots, T^{(m-1)\ell}(g_r)$ span $\mathcal{C}$. If we denote by $g_{i,j}$ the j'th coordinate of g_i and let*

$$G_i = \begin{pmatrix} g_{1,i\ell+1} & \cdots & g_{1,(i+1)\ell} \\ \vdots & & \vdots \\ g_{r,i\ell+1} & \cdots & g_{r,(i+1)\ell} \\ & 0 & \end{pmatrix} \in M_\ell(\mathbb{F}_q)$$

and

$$g(X) = \frac{1}{X^\nu} \sum_{i=0}^{m-1} G_i X^i \in M_\ell(\mathbb{F}_q)[X],$$

where ν is the least integer such that $G_i \neq 0$, then $\mathcal{C}$ corresponds to the left ideal $\langle g(X) \rangle$ by Theorem 4.

Corollary 225. *Taking the notation of the proof of Theorem 4, the submodule* $\varphi(\mathcal{C}) \subseteq (\mathbb{F}_q[X]/(X^m-1))^\ell$ *is generated by* r *elements as an* $\mathbb{F}_q[X]/(X^m-1)$*-module but cannot be generated by less that* r *elements. If* $\mathcal{C}$ *is a cyclic code then we have* $r = 1$ *and we find the classical result about cyclic codes.*

Definition 226 (Generator polynomial)**.** The polynomial $g(X) \in M_\ell(\mathbb{F}_q)[X]$ from Corollary 224 is called a *generator polynomial* of $\mathcal{C}$.

Example 227. Let $\mathbb{F}_4 = \mathbb{F}_2[\omega]$ and $I = \langle P(X), Q(X)\rangle \subset M_3(\mathbb{F}_4)[X]/(X^5-1)$ be a left ideal with

$$P(X) = \begin{pmatrix} \omega & 0 & 1 \\ \omega & \omega & 0 \\ \omega^2 & \omega^2 & 0 \end{pmatrix} X^4 + \begin{pmatrix} \omega & \omega^2 & \omega^2 \\ 0 & \omega & 1 \\ \omega^2 & 0 & \omega \end{pmatrix} X^3 + \begin{pmatrix} 0 & \omega^2 & \omega \\ \omega & \omega^2 & \omega^2 \\ 0 & 1 & \omega^2 \end{pmatrix} X^2 + \begin{pmatrix} 1 & 0 & \omega^2 \\ 0 & \omega & 1 \\ 0 & \omega & 1 \end{pmatrix} X + \begin{pmatrix} 1 & 0 & \omega^2 \\ 0 & 1 & \omega^2 \\ 0 & 0 & 0 \end{pmatrix}$$

and

$$Q(X) = \begin{pmatrix} \omega & 0 & 1 \\ \omega & \omega & 0 \\ \omega^2 & \omega^2 & 0 \end{pmatrix} X^4 + \begin{pmatrix} 0 & 1 & \omega^2 \\ 0 & 1 & \omega^2 \\ 0 & \omega & 1 \end{pmatrix} X^3 + \begin{pmatrix} \omega & \omega^2 & \omega^2 \\ 1 & \omega & \omega \\ \omega & \omega^2 & \omega^2 \end{pmatrix} X^2 + \begin{pmatrix} 1 & \omega^2 & 1 \\ 0 & \omega^2 & \omega \\ 0 & 1 & \omega^2 \end{pmatrix} X + \begin{pmatrix} 1 & 1 & 0 \\ \omega & \omega^2 & \omega^2 \\ \omega^2 & 1 & 1 \end{pmatrix}.$$

The row echelon form generator matrix of the 3-quasi cyclic code $\mathcal{C}_I$ associated to the left ideal I is

$$G = \left(\begin{array}{ccc|ccc|ccc|ccc|ccc} 1 & 0 & \omega^2 & 0 & 0 & 0 & 0 & \omega^2 & \omega & \omega & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & \omega^2 & 0 & 0 & 0 & 0 & 0 & 0 & \omega & \omega & 0 & 1 & 0 & \omega^2 \\ \hline 0 & 0 & 0 & 1 & 0 & \omega^2 & 0 & 0 & 0 & 0 & \omega^2 & \omega & \omega & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & \omega^2 & 0 & \omega^2 & \omega & \omega & 0 & 1 & \omega & \omega & 0 \\ \hline 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & \omega^2 & 0 & \omega & 0 & \omega^2 & \omega \end{array}\right).$$

Algorithm 27 gives that $(g_4, g_5, T^3(g_4), T^3(g_5), T^{2\times 3}(g_5))$ is a basis of $\mathcal{C}_I$. Moreover

$$g(X) = \begin{pmatrix} 0 & 1 & \omega^2 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix} + \begin{pmatrix} 0 & \omega^2 & \omega \\ 1 & 1 & 0 \\ 0 & 0 & 0 \end{pmatrix} X + \begin{pmatrix} \omega & 0 & 1 \\ \omega^2 & 0 & \omega \\ 0 & 0 & 0 \end{pmatrix} X^2 + \begin{pmatrix} \omega & \omega & 0 \\ 0 & \omega^2 & \omega \\ 0 & 0 & 0 \end{pmatrix} X^3$$

is a generator polynomial of $\mathcal{C}_I$ and $I = \langle P(X), Q(X)\rangle = \langle g(X)\rangle$.

5.2.3 A property of generator polynomials

The following proposition generalizes [MS86a, Theorem 1, (c), page 190] and [MS86a, Theorem 4, page 196].

Proposition 228. *Let $\mathcal{C}$ be an ℓ-quasi-cyclic code of length $m\ell$ over $\mathbb{F}_q$. Let $P(X)$ be a generator polynomial of $\mathcal{C}$ and $Q(X)$ a generator polynomial of its dual. Then*

$$P(X)\left({}^tQ^\star(X)\right) \equiv 0 \pmod{X^m - 1}$$

where $Q^\star(X) = X^{\deg(Q)}Q(1/X)$ denotes the reciprocal polynomial of Q and ${}^tQ$ the polynomial whose coefficients are the transposed matrices of the coefficients of Q.

Proof. Since $P(X) = \sum_{i=0}^{m-1} P_i X^i$ is a generator polynomial of $\mathcal{C}$, the rows of the matrix

$$\begin{pmatrix} P_0 & P_1 & \dots & P_{m-1} \end{pmatrix}$$

and their shifts span $\mathcal{C}$. Similarly $Q(X) = \sum_{i=0}^{m-1} Q_i X^i$ and the rows of

$$\begin{pmatrix} Q_0 & Q_1 & \dots & Q_{m-1} \end{pmatrix}$$

and their shifts span $\mathcal{C}^\perp$. By definition of a dual code, we have

$$\begin{pmatrix} P_0 & P_1 & \cdots & P_{m-1} \end{pmatrix} \begin{pmatrix} {}^tQ_0 \\ {}^tQ_1 \\ \vdots \\ {}^tQ_{m-1} \end{pmatrix} = \sum_{i=0}^{m-1} P_i \left({}^tQ_i\right) = 0.$$

As $\mathcal{C}$ and $\mathcal{C}^\perp$ are ℓ-quasi cyclic codes we also have

$$\sum_{i=0}^{m-1} P_i \left({}^tQ_{i+j \mod m}\right) = 0$$

for all $j \in \mathbb{Z}$. Therefore

$$P(X)\left({}^tQ^\star(X)\right) = \sum_{j=0}^{m-1}\sum_{i=0}^{m-1} P_i \left({}^tQ_{i-j \mod m}\right) X^j = 0 \mod (X^m - 1).$$

Hence the proposition. □

5.3 Quasi-BCH

In Section 5.2 we saw that quasi-cyclic codes can be regarded as a generalization of cyclic codes. Therefore, it is interesting to focus on the generalization of BCH codes. We start with the definition and then study their parameters. Finally we present a decoding scheme for quasi-BCH codes raising interesting questions. We fix four positive integers $n = m\ell$ and s.

5.3.1 Definition

Definition 229 (Primitive root of unity)**.** Let q be a prime power. A matrix $A \in M_\ell(\mathbb{F}_{q^s})$ is called a *primitive m-th root of unity* if

- $A^m = I_\ell$,
- $A^i \neq I_\ell$ if $i < m$,
- $\det(A^i - A^j) \neq 0$, whenever $i \neq j$.

Proposition 230. *Let q be a prime power and suppose that $q^{s\ell} - 1 = m$. Then there exists a primitive m-th root of unity in $M_\ell(\mathbb{F}_{q^s})$.*

Proof. Let $\alpha \in \mathbb{F}_{q^{s\ell}}$ be a primitive m-th root of unity and $A \in M_\ell(\mathbb{F}_{q^s})$ be the companion matrix of the irreducible polynomial $f(X) \in \mathbb{F}_{q^s}[X]$ of α over $\mathbb{F}_{q^s}$. There exists $P \in \mathrm{GL}_\ell(\mathbb{F}_{q^{s\ell}})$ and an upper triangular matrix $U \in M_\ell(\mathbb{F}_{q^{s\ell}})$ whose diagonal coefficients are the eigenvalues of A such that $A = P^{-1}UP$. The eigenvalues of A are exactly the roots of f and then are primitive m-th roots of unity. Therefore A satisfies the three conditions of Definition 229. □

Definition 231 (Block minimum distance)**.** Let $\mathcal{C}$ be a linear code over $\mathbb{F}_q$ of length $m\ell$. We define the *ℓ-block minimum distance* of $\mathcal{C}$ to be the minimum distance of the folded code of $\mathcal{C}$.

Definition 232 (Left quasi-BCH codes)**.** Let A be a primitive m-th root of unity in $M_\ell(\mathbb{F}_{q^s})$ and $\delta \leq m$. We define the ℓ-quasi-BCH code of length $m\ell$, with respect to A, with designed minimum distance δ, over $\mathbb{F}_q$ by

$$\text{Q-BCH}_q(m, \ell, \delta, A) := \left\{ (c_1, \ldots, c_m) \in (\mathbb{F}_q^\ell)^m : \sum_{j=0}^{m-1} A^{ij} c_{j+1} = 0 \text{ for } i = 1, \ldots, \delta - 1 \right\}.$$

We call the linear map

$$\begin{array}{rcl} \mathcal{S}_A : (\mathbb{F}_q^\ell)^m & \to & (\mathbb{F}_{q^s}^\ell)^m \\ x = (x_1, \ldots, x_m) & \mapsto & \sum_{j=0}^{m-1} A^j x_{j+1} \end{array}$$

the *syndrome* map with respect to Q-BCH(m, ℓ, δ, A).

Proposition 233. *Using the notation of Definition 232,* Q-BCH$_q(m, \ell, \delta, A)$ *has dimension at least $(m - s(\delta - 1))\ell$ and ℓ-block minimum distance at least δ. In other words* Q-BCH$_q(m, \ell, \delta, A)$ *is an $[m\ell, \geq (m - s(\delta - 1))\ell, \geq \delta]_{\mathbb{F}_q}$-code.*

Proof. According to Definition 232 we have that

$$H = \begin{pmatrix} I_\ell & A & \cdots & A^{m-1} \\ I_\ell & A^2 & \cdots & A^{2(m-1)} \\ \vdots & \vdots & & \vdots \\ I_\ell & A^{\delta-1} & \cdots & A^{(\delta-1)(m-1)} \end{pmatrix} \in M_{(\delta-1)\ell, m\ell}(\mathbb{F}_{q^s})$$

is a parity check matrix of Q-BCH$_q(m, \ell, \delta, A)$. Let

$$V = \begin{pmatrix} I_\ell & A & \cdots & A^{\delta-1} \\ I_\ell & A^2 & \cdots & A^{2(m-1)} \\ \vdots & \vdots & & \vdots \\ I_\ell & A^{\delta-1} & \cdots & A^{(\delta-1)^2} \end{pmatrix}.$$

Using the Vandermonde matrix trick we find that the determinant D of V over $M_\ell(\mathbb{F}_{q^s})[A]$ is $\prod_{i<j}(A^i - A^j)$. By the definition of A we have $\det_{\mathbb{F}_{q^s}} D \neq 0$, thus V is invertible over $M_\ell(\mathbb{F}_{q^s})[A]$ and then, invertible over $\mathbb{F}_{q^s}$. Therefore H has full rank over $\mathbb{F}_{q^s}$.

Let $i : \mathbb{F}_q^{m\ell} \to \mathbb{F}_{q^s}^{m\ell}$ be the canonical injection and denote by $h : \mathbb{F}_{q^s}^{m\ell} \to \mathbb{F}_{q^s}^{(\delta-1)\ell}$ the $\mathbb{F}_q$-linear map given by H. Then we have $\dim_{\mathbb{F}_q}(\operatorname{Im} h) = s(\delta-1)\ell$. Thus $\dim_{\mathbb{F}_{q^s}}(\operatorname{Im} h \circ i) \leq (\delta-1)\ell$ and $\dim_{\mathbb{F}_q}(\operatorname{Im} h \circ i) \leq s(\delta-1)\ell$. Therefore $\dim_{\mathbb{F}_q}(\ker h \circ i) \geq m\ell - s(\delta-1)\ell$. Suppose that there exists a codeword $c = (c_1, \ldots, c_m) \in \mathcal{C} \setminus \{0\}$ with ℓ-block weight $b \leq \delta - 1$. Note $i_1, \ldots, i_b$ the indexes such that $c_{i_j} \neq 0$ for $i = 1, \ldots, b$. This implies that the matrix

$$\begin{pmatrix} A^{i_1} & A^{i_2} & \cdots & A^{i_b} \\ A^{2i_1} & A^{2i_2} & \cdots & A^{2i_b} \\ \vdots & \vdots & & \vdots \\ A^{(\delta-1)i_1} & A^{(\delta-1)i_2} & \cdots & A^{(\delta-1)i_b} \end{pmatrix}$$

has not full rank which is absurd. □

Example 234. Consider the 3-quasi-BCH codes defined by primitive roots in $M_3(\mathbb{F}_{2^2})$ of length 63 over $\mathbb{F}_2$ with designed minimum distance 6 defined by a 21-th root of unity in $\mathbb{F}_{2^6}$. In other words, $q = 2, m = 21, \ell = 3, s = 2$ and $\delta = 6$. There are 22 non-equivalent codes splitting as follows:

Number of codes	Parameters
2	$[63, 33, 6]_{\mathbb{F}_2}$
18	$[63, 33, 7]_{\mathbb{F}_2}$
2	$[63, 36, 6]_{\mathbb{F}_2}$

Notice that their dimension is always at least $(m - s(\delta-1))\ell = 33$ and their minimum distance is at least $\delta = 6$. All the computations have been performed with the MAGMA computer algebra system [BCP97].

Example 235. Let $q = 5, m = 7, \ell = 3, s = 2$ and $\delta = 3$. Let $\omega \in \mathbb{F}_{5^2}$ be a primitive $(5^2 - 1)$-th root of unity and

$$A = \begin{pmatrix} \omega^9 & \omega^4 & \omega^{22} \\ \omega^{11} & \omega^{11} & \omega^{15} \\ \omega^2 & \omega^{19} & 1 \end{pmatrix} \in M_3(\mathbb{F}_{5^2}).$$

Then the left 3-quasi-BCH code of length 21 with respect to A with designed minimum distance 3 over $\mathbb{F}_5$ has parameters $[21, 9, 7]_{\mathbb{F}_5}$. Its generator polynomial is given by

$$g(X) = \begin{pmatrix} 1 & 4 & 3 \\ 3 & 3 & 4 \\ 1 & 1 & 4 \end{pmatrix} X^4 + \begin{pmatrix} 4 & 0 & 0 \\ 4 & 0 & 0 \\ 4 & 0 & 4 \end{pmatrix} X^3 + \begin{pmatrix} 3 & 0 & 4 \\ 0 & 3 & 4 \\ 0 & 0 & 0 \end{pmatrix} X^2 + \begin{pmatrix} 2 & 3 & 2 \\ 4 & 4 & 4 \\ 3 & 1 & 1 \end{pmatrix} X + \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \in M_3(\mathbb{F}_5)[X].$$

5.4 Decoding scheme for quasi-BCH codes

For this section we fix five positive integers $n = m\ell$, r and δ, a primitive m-th root of unity $A \in M_\ell(\mathbb{F}_{q^s})$ and $\mathcal{C} = \text{Q-BCH}(m, \ell, \delta, A)$. If the folded of $\mathcal{C}$ is a BCH code $\mathcal{C}'$ over $\mathbb{F}_{q^\ell}$ (which is not the case in general) then we can apply the standard, unique and list, decoding algorithms. See for example [MS86a, Paragraph 6, page 270] and [ABC11]. If $\mathcal{C}'$ is not a code for which a decoding algorithm is known, we propose in what follows a decoding scheme for $\mathcal{C}$ based on the key equation that we establish for quasi-BCH codes. Following the same techniques as for BCH codes, we first compute the locator and evaluator polynomials by solving the key equation and then compute the error vector and recover the original message.

Notation 236. Let κ be any field and $x = (x_1, \ldots, x_n) \in \kappa^n$. We denote by $w(x)$ the Hamming weight of x *i.e.* the cardinal of $W = \{i : i \in \{1, \ldots, n\} \text{ s.t. } x_i \neq 0\}$. We denote by $\text{Supp}(x)$ the support of x *i.e.* the set W.

5.4.1 The key equation

As in the scalar case, we exhibit a key equation for quasi-BCH codes. In this subsection, all vectors are considered to be single-column matrices. Consider $\mathbb{F}_q^\ell$ as a product ring of ℓ copies of $\mathbb{F}_q$. We define a map

$$\begin{array}{rcl} \Psi : M_\ell(\mathbb{F}_{q^s})[[X]] \times \mathbb{F}_q^\ell[[X]] & \to & \mathbb{F}_{q^s}^\ell[[X]] \\ (f, g) & \mapsto & \sum_{i,j} f_j g_i X^{i+j} \end{array}$$

where the $f_i g_j$ are matrix-vector products. In the sequel we will denote $\Psi(f, g)$ simply by $f \diamond g$. Note that we have $(fh) \diamond g = f \diamond (h \diamond g)$ for any $h \in M_\ell(\mathbb{F}_{q^s})$.

Let c be a codeword of $\mathcal{C}$ sent over a channel, $y \in (\mathbb{F}_q^\ell)^m$ be the received word and let e be the error vector *i.e.* $e = y - c$ such that $\mathrm{w}(e) = w \leq \lfloor (\delta - 1)/2 \rfloor$. Let $W = \mathrm{Supp}(e) = \{i_1, \ldots, i_w\}$.

Definition 237 (Locator and evaluator polynomials)**.** We define the *locator polynomial* by

$$\Lambda(X) := \prod_{i \in W} (1 - A^i X) \in M_\ell(\mathbb{F}_{q^s})$$

and the *evaluator polynomial* by

$$L(X) := \sum_{i \in W} \left(\prod_{j \neq i}^{w} A^i (1 - A^j) X \right) \diamond y_i \in \mathbb{F}_{q^s}^\ell [X].$$

Lemma 238. *Let $B \in M_\ell(\mathbb{F}_q)$ be a nonzero matrix, then $1 - BX$ has a left- and right-inverse in $M_\ell(\mathbb{F}_q)[[X]]$, both equal to*

$$\sum_{j=0}^{+\infty} B^j X^j.$$

We see that the locator polynomial $\Lambda(X)$ is invertible in the power series ring $M_\ell(\mathbb{F}_{q^s})[[X]]$ and we have

$$\begin{aligned} \left(\Lambda(X)^{-1}\right) \diamond L(X) &= \sum_{i \in W} \left(A^i (1 - A^i X)^{-1} \right) \diamond y_i \\ &= \sum_{i \in W} \left(\sum_{j=0}^{+\infty} A^{i(j+1)} X^j \right) \diamond y_i \\ &= \sum_{j=0}^{+\infty} \sum_{i \in W} A^{i(j+1)} y_i X^j. \end{aligned}$$

Using the fact that $y = c + e$ and that, by definition, $\mathcal{S}_{A^i}(y) = \mathcal{S}_{A^i}(e)$ for any $i = 0, \ldots, \delta - 1$ we have

$$\left(\Lambda(X)^{-1}\right) \diamond L(X) = \sum_{j=0}^{+\infty} \mathcal{S}_{A^{j+1}}(e) X^j := S_\infty(X).$$

Proposition 239. *For any error vector $e \in \mathbb{F}_q^{m\ell}$ such that $w(e) \leq \lfloor (\delta - 1)/2 \rfloor$ we have*

$$\boxed{\Lambda(X) \diamond S_\infty(X) = L(X)}$$

and therefore

$$\Lambda(X) \diamond S_\infty(X) \equiv L(X) \mod X^\delta. \tag{5.2}$$

We will refer to 5.2 as the key equation*.*

Problems solving the key equation

In the case of BCH codes, the extended Euclidean and Berlekamp-Massey algorithms can be used to solve the key equation. We denote by $S_\delta(X)$ the polynomial $S_\infty(X)$ mod X^δ from 5.2 which can be written as

$$\begin{pmatrix} \Lambda_0 & \dots & \Lambda_{\delta-1} & \vert & L_0 & \dots & L_{\delta-1} \end{pmatrix} \left(\begin{array}{cccc} S_0 & S_1 & \dots & S_{\delta-1} \\ & S_0 & & \vdots \\ & & \ddots & \vdots \\ & & & S_0 \\ \hline -1 & 0 & \dots & 0 \\ 0 & -1 & & \vdots \\ \vdots & & \ddots & 0 \\ 0 & \dots & 0 & -1 \end{array}\right) = 0. \tag{5.3}$$

Where the S_i's and L_i's are column vectors such that the S_i's are the coefficients of S_δ in $\mathbb{F}_{q^s}^\ell$ and the L_i's are the coefficients in $\mathbb{F}_{q^s}^\ell$ of $L(X)$. The Λ_i's are the coefficients of $\Lambda(X)$ in $M_\ell(\mathbb{F}_{q^s})$. This system of linear equations over $\mathbb{F}_{q^s}$ has many solutions in $\mathbb{F}_{q^s}$ since there are $\ell\delta+\delta$ unknowns and only δ equations for each row of

$$\begin{pmatrix} \Lambda_0 & \dots & \Lambda_{\delta-1} & \vert & L_0 & \dots & L_{\delta-1} \end{pmatrix}.$$

However, we are only interested in the solution such that $(\Lambda_0, \dots, \Lambda_{\delta-1})$ is an error locator polynomial. In other words, if we let $\mathfrak{B}$ be the solutions of 5.3 and

$$\mathfrak{S} = \left\{ \prod_{i\in W} (1 - A^i X) \in M_\ell(\mathbb{F}_{q^s}) : W \subset \{1, \dots, m\} \text{ and } \#W \leq \lfloor (\delta-1)/2 \rfloor \right\}$$

be the set of all possible locator polynomials corresponding to errors of weight at most $\lfloor (\delta-1)/2 \rfloor$, we are interested in the elements of $\mathfrak{B} \cap \mathfrak{S}$.

Proposition 240. *There exists one and only one solution of equation 5.3 in* $\mathfrak{S}$.

Proof. Equation 5.2 ensures that there exists at least one element in $\mathfrak{B} \cap \mathfrak{S}$. If there were more than one solution in $\mathfrak{S}$ there would exist more than one codeword in a Hamming ball of radius $\lfloor (\delta-1)/2 \rfloor$ which is absurd. □

The solving of 5.3 remains difficult. One needs an exponential (in $\ell\delta$) number of arithmetic operations in $\mathbb{F}_{q^s}$ to find the element of $\mathfrak{B} \cap \mathfrak{S}$. For small values of q, ℓ and δ the solution can be found by exhaustive search on the solutions of 5.3.

Unambiguous decoding scheme

In this subsection, we prove that, as in the BCH case, the roots of the locator polynomial (in $\mathbb{F}_{q^s}[A]$) give precious information about the location of errors. The factorization of polynomials of $M_\ell(\mathbb{F}_{q^s})[X]$ is not unique, all the roots of the locator polynomial do not indicate an error position.

Proposition 241. *Let $e \in \mathbb{F}_q^{m\ell}$ be an error vector such that $w(e) \leq \lfloor (\delta-1)/2 \rfloor$ and $\Lambda(X)$ be the locator polynomial associated to e. We have*

$$e_i \neq 0 \Longleftrightarrow \Lambda(A^{-i}) = 0.$$

Proof. By definition, we have $\Lambda(A^{-i}) = 0$ if $e_i \neq 0$. Conversely, if $e_i = 0$ then $A^j A^{-i} \neq I_\ell$ for $j \in \mathrm{Supp}(e)$. Thus $1 - A^j A^{-i}$ is a unit in $\mathbb{F}_{q^s}[A]$ by definition of A. Therefore $\Lambda(A^{-i}) \neq 0$. □

These roots can be found by an exhaustive search on the powers of A in at most m attempts. At this step the support of the error vector e is known. The last step to complete the decoding is to find the value of the error.

Proposition 242. *Let $e \in \mathbb{F}_q^{m\ell}$ be an error such that $w(e) \leq \lfloor (\delta-1)/2 \rfloor$, $W = \mathrm{Supp}(e)$, $\Lambda(X)$ be the locator and $L(X)$ be the evaluator polynomials associated to e. If A^{-i} is a root of $\Lambda(X)$ for $i \in W$, then*

$$e_i = \prod_{j \in W \setminus \{i\}} (A^i - A^j)^{-1} L(A^{-i})$$

where $L(A^j)$ denotes $\sum (A^j)^i L_i$.

Proof. Let $i_0 \in W$. We have

$$\begin{aligned} L(A^{-i_0}) &= \sum_{i=1}^{w} \prod_{j \neq i}^{w} A_i (1 - A^{-i_0} A_j) y_i \\ &= \prod_{j \in W \setminus \{i_0\}} A^{i_0} (1 - A^{-i_0} A^j) e_{i_0} \\ &= \prod_{j \in W \setminus \{i_0\}} (A^{i_0} - A^j) e_{i_0}. \end{aligned}$$

By definition of A, $A^{i_0} - A^j$ is invertible for all $j \in W$ hence the result. □

5.5 Evaluation codes

5.5.1 Definition and parameters

In this subsection we generalize evaluation codes. For any ring R and any positive integer k, we denote by $R[X]_{<k}$ the left R-module of all polynomials of $R[X]$ of degree at most $k-1$.

Proposition 243. *Let q be a prime power and ℓ, m be positive integers such that $m = q^\ell - 1$. Let $A \in M_\ell(\mathbb{F}_q)$ be a primitive m-th root of unity. Then $\mathbb{F}_q[A]$ and $\mathbb{F}_{q^\ell}$ are isomorphic as rings.*

Algorithm 28 Decoding algorithm for quasi-BCH codes

Input: The received word $y = c + e$ where $c \in \mathcal{C}$ and $w(e) \leq \lfloor (\delta - 1)/2 \rfloor$.
Output: The codeword c, if it exists such that $d(y, c) \leq \lfloor (\delta - 1)/2 \rfloor$.

$S_\delta(X) \leftarrow$ Syndrome of y.
Compute $\Lambda(X)$ and $L(X)$ (Subsection 5.4.1).
$\mathfrak{R} \leftarrow$ roots of $\Lambda(X)$ in $\mathbb{F}_{q^s}[A]$.
$W \leftarrow \{i | A^{-i} \in \mathfrak{R}\}$.
$\zeta \leftarrow (0, \ldots, 0)$.
for $i \in W$ **do**
  $\zeta_i = \prod_{j \in W \setminus \{i\}} (A^i - A^j)^{-1} L(A^{-i})$.
end for
return $y - \zeta$.

Proof. Let $\mu(X)$ be the minimal polynomial of A of degree at most ℓ. We have $\mu | X^m - 1$, thus the roots of μ are all distinct. By Definition 229-(3), the roots of μ lie in $\mathbb{F}_{q^\ell}$ and not in any subfield. Therefore μ is irreducible. □

Definition 244 (Quasi-cyclic evaluation codes)**.** Let ℓ be a positive integer and q be a prime power. Let $m = q^\ell - 1$ and $k \leq m$. Let $A \in M_\ell(\mathbb{F}_q)$ a primitive m-th root of unity. Let π be a $\mathbb{F}_q$-linear map from $\mathbb{F}_q[A]$ into $\mathbb{F}_q^\ell$. We denote by $C_{A,k,\pi}$ the image of:

$$\begin{array}{ccccccc} (\mathbb{F}_q[A])[X]_{<k} & \xrightarrow{\mathrm{ev}_A} & (\mathbb{F}_q[A])^m & \xrightarrow{\pi^{\times m}} & (\mathbb{F}_q^\ell)^m \\ P(X) & \longmapsto & \left(P(A^0), \ldots, P(A^{m-1})\right) & \longmapsto & \left(\pi(P(A^0)), \ldots, \pi(P(A^{m-1}))\right). \end{array}$$

Proposition 245. *Taking the notation of Definition 244, $C_{A,k,\pi}$ is a ℓ-quasi cyclic code over $\mathbb{F}_q$ of length $m\ell$ and of dimension over $\mathbb{F}_q$ at least $k\ell - \dim_{\mathbb{F}_q}(\ker \pi^{\times m})$.*

Proof. By Proposition 243 the statement about the dimension of $C_{A,k,\pi}$ is obvious. Let

$$P(X) = \sum_{i=0}^{k-1} \sum_{j=0}^{m-1} P_{ij} A^j X^i \in \mathbb{F}_q[A][X]_{<k}$$

with $P_{ij} \in \mathbb{F}_q$. Then

$$Q(X) = \sum_{i=0}^{k-1} \sum_{j=0}^{m-1} P_{ij} A^{j+i} X^i \in \mathbb{F}_q[A][X]_{<k}$$

is such that $Q(A^i) = P(A^{i+1})$ for all $i \in \mathbb{Z}$ and $C_{A,k,\pi}$ is ℓ-quasi cyclic. □

5.5.2 New good codes

Proposition 246. *Using the notation of Definition 244, if π is such that for $B = (b_{ij}) \in \mathbb{F}_q[A]$*

- $\pi(B) = (b_{i1}, \ldots, b_{i\ell})$ *for some* i,
- *or* $\pi(B) = (b_{1j}, \ldots, b_{\ell j})$ *for some* j,

then $\dim C_{A,k,\pi} \geq k\ell$ *and* $C_{A,k,\pi}$ *has minimum distance* $d \geq m - k + 1$.

Proof. In both cases, it suffices to notice that $\pi^{\times m}$ is injective. If $\pi^{\times m}(B_1, \ldots, B_m) = 0$ then $\det B_i = 0$ for $i = 1, \ldots, m$. As $\mathbb{F}_q[A]$ is a field we must have $B_i = 0$ for $i = 1, \ldots, m$. In fact under the assumptions of the proposition $\pi^{\times m}$ is an isomorphism since $\#((F_q[A])^m) = q^{m\ell} = \#((F_q^\ell)^m)$. □

Remark 247. All the computations of the examples below have been performed with the MAGMA computer algebra system [BCP97].

1. For some particular choices of π, especially when we decrease the dimension k, we observe that the minimum distance is multiplied by $\ell - 1$. For example, with

$$A = \begin{pmatrix} 0 & \omega & 0 \\ \omega & \omega^2 & \omega^2 \\ 1 & \omega^2 & 1 \end{pmatrix} \in M_3(\mathbb{F}_4) \text{ with } \mathbb{F}_4 = \mathbb{F}_2[\omega],$$

   $k = 4$ and $\pi((b_{ij})) = (b_{2,1}, b_{1,2}, b_{2,3})$, we find a $[189, 11, 125]_{\mathbb{F}_4}$-code. According to [Gra07], the previous best known minimum distance was 121.

2. As for Reed-Solomon codes, we can evaluate polynomials of $(\mathbb{F}_q[A])[X]_{<k}$ at less than $m = q^\ell - 1$ points. Following this approach, we find the following new good codes listed below together with the corresponding previous best known minimum distances:

$$[186, 11, 122]_{\mathbb{F}_4},\ 120;$$
$$[183, 11, 119]_{\mathbb{F}_4},\ 117;$$
$$[180, 11, 116]_{\mathbb{F}_4},\ 114;$$
$$[177, 11, 113]_{\mathbb{F}_4},\ 112.$$

3. Markus Grassl applied different methods to construct new codes from our $[189, 11, 125]_{\mathbb{F}_4}$ code (item 1 of Remark 247). For example, he used a puncturing method [GW04]. Some of the codes he obtained have the same parameters as the codes listed in item 2 of Remark 247. He found $[186, 11, 122]_{\mathbb{F}_4}$, $[183, 11, 119]_{\mathbb{F}_4}$ and $[180, 11, 116]_{\mathbb{F}_4}$ codes. He also found a $[177, 11, 114]_{\mathbb{F}_4}$ code while the best known minimum distance was 112. The 49 new codes found with the help of Markus Grassl are listed in Table 5.1. All the methods used for the construction of these codes are detailed in [Gra07].

Remark 248. We have proved in Proposition 243 that $\mathbb{F}_q[A]$ is a field such that $[\mathbb{F}_q[A] : \mathbb{F}_q] = \ell$. Thus there is a $\mathbb{F}_q$-linear isomorphism from $\mathbb{F}_q[A]$ to $\mathbb{F}_q^\ell$. Consider the following one:

$$\begin{array}{ccc} \mathbb{F}_q[A] & \xrightarrow{\psi} & \mathbb{F}_q^\ell \\ B = b_0 I_\ell + b_1 A + \cdots + b_{\ell-1} A^{\ell-1} & \longmapsto & (b_0, b_1, \ldots, b_{\ell-1}). \end{array}$$

New codes over $\mathbb{F}_4$				
$[171, 11, 109]_4$	$[172, 11, 110]_4$	$[173, 11, 110]_4$	$[174, 11, 111]_4$	$[175, 11, 112]_4$
$[176, 11, 113]_4$	$[177, 11, 114]_4$	$[178, 11, 115]_4$	$[179, 11, 115]_4$	$[180, 11, 116]_4$
$[181, 11, 117]_4$	$[182, 11, 118]_4$	$[183, 11, 119]_4$	$[184, 10, 121]_4$	$[184, 11, 120]_4$
$[185, 10, 122]_4$	$[185, 11, 121]_4$	$[186, 10, 123]_4$	$[186, 11, 122]_4$	$[187, 10, 124]_4$
$[187, 11, 123]_4$	$[188, 10, 125]_4$	$[188, 11, 124]_4$	$[189, 10, 126]_4$	$[189, 11, 125]_4$
$[190, 10, 127]_4$	$[190, 11, 126]_4$	$[191, 10, 128]_4$	$[191, 11, 127]_4$	$[192, 11, 128]_4$
$[193, 11, 128]_4$	$[194, 11, 128]_4$	$[195, 11, 128]_4$	$[196, 11, 129]_4$	$[197, 11, 130]_4$
$[198, 11, 130]_4$	$[199, 11, 131]_4$	$[200, 11, 132]_4$	$[201, 10, 133]_4$	$[201, 11, 132]_4$
$[202, 10, 134]_4$	$[202, 11, 132]_4$	$[203, 10, 135]_4$	$[204, 10, 136]_4$	$[204, 11, 133]_4$
$[205, 11, 134]_4$	$[210, 11, 137]_4$	$[213, 11, 139]_4$	$[214, 11, 140]_4$	

Table 5.1: 49 new codes over $\mathbb{F}_4$ which have a larger minimum distance than the previously known ones.

Then

$$C_{A,k,\psi} = \psi^{\times m}(\mathrm{ev}_A(\mathbb{F}_q[A][X]_{<k}))$$

is still an ℓ-quasi cyclic code of length $m\ell$ and of dimension $k\ell$. Let $\Pi \in M_\ell(\mathbb{F}_q)$ and let

$$\begin{array}{rcl} \pi : \mathbb{F}_q^\ell & \to & \mathbb{F}_q^\ell \\ x & \mapsto & x\Pi \end{array}$$

for a given $\Pi \in M_\ell(\mathbb{F}_q)$. Then

$$C_{A,k,\psi,\pi} = \pi^{\times m}(\psi^{\times m}(\mathrm{ev}_A(\mathbb{F}_q[A][X]_{<k})))$$

is an ℓ-quasi cyclic code of length $m\ell$ and dimension $\geq k\ell - \dim(\ker \pi)$.

We notice that there exist matrices Π for which the obtained minimum distance is always greater than $m - k + 1$. For instance, taking $\ell = 3$, $q = 4$ and the matrix

$$\Pi = \begin{pmatrix} 1 & \omega^2 & \omega \\ \omega^2 & \omega & 1 \\ 1 & 1 & 1 \end{pmatrix},$$

give codes with minimum distance close to $2(m - k + 1)$.

5.6 Conclusion

In this paper we presented a generalization of results for cyclic codes to quasi-cyclic codes. We proved that there is a natural one-to-one correspondence between ℓ-quasi-cyclic codes and left ideals of $M_\ell(\mathbb{F}_q)[X]/(X^m - 1)$. We then extended the construction of BCH and evaluation codes to this context. This generalization allowed us to find a lot of new codes with good parameters and, sometimes, beating previous known minimum distances. A deeper study of decoding algorithms for quasi-BCH need more work and remains an open problem.

Acknowledgments

We would like to thank the referees, whose suggestions have permit to improve this article, in particular, for Algorithm 27 and for the idea of using the Morita equivalence to prove the one-to-one correspondence between left ideals and quasi-cyclic codes. We would like to thank Markus Grassl for his precious help for finding new good codes.

Chapter 6

An algorithm for list decoding number field codes

This chapter constitutes an accepted paper at ISIT (International Symposium on Information Theory) 2012. It has been done in collaboration with Jean-François Biasse.

Abstract—We present an algorithm for list decoding codewords of algebraic number field codes in polynomial time. This is the first explicit procedure for decoding number field codes whose construction were previously described by Lenstra [Len86] and Guruswami [Gur03]. We rely on a new algorithm for computing the Hermite normal form of the basis of an $\mathcal{O}_K$-module due to Biasse and Fieker [BF12] where $\mathcal{O}_K$ is the ring of integers of a number field K.

6.1 Introduction

Algorithms for list decoding Reed-Solomon codes, and their generalization the algebraic-geometric codes are now well understood. The codewords consist of sets of functions whose evaluation at a certain number of points are sent, thus allowing the receiver to retrieve them provided that the number of errors is manageable.

The idea behind algebraic-geometric codes can be adapted to define algebraic codes whose messages are encoded as a list of residues redundant enough to allow errors during the transmission. The Chinese Remainder codes (CRT codes) have been fairly studied by the community [GSS00, Man76]. The encoded messages are residues modulo $N := p_1, \cdots, p_n$ of numbers $m \leq K := p_1 \cdots p_k$ where $p_1 < p_2 < \cdots < p_n$ are prime numbers. They are encoded by using

$$\begin{array}{ccc} \mathbb{Z} & \longrightarrow & \mathbb{Z}/p_1 \times \cdots \times \mathbb{Z}/p_n \\ m & \longmapsto & (m \bmod p_1, \cdots, m \bmod p_n). \end{array}$$

Decoding algorithms for CRT codes were significantly improved to reach the same level of tolerance to errors as those for Reed-Solomon codes [Bon00, GRS99, GSS00]. As algebraic-geometric codes are a generalization of Reed-Solomon codes, the idea arose

that we could generalize the results for CRT codes to redundant residue codes based on number fields. Indeed, we can easily define an analogue of the CRT codes where a number field K plays the role of $\mathbb{Q}$ and its ring of integers $\mathcal{O}_K$ plays the role of $\mathbb{Z}$. Then, for prime ideals $\mathfrak{p}_1, \cdots, \mathfrak{p}_n$ such that $\mathcal{N}(\mathfrak{p}_1) < \cdots < \mathcal{N}(\mathfrak{p}_n)$, a message $m \in \mathcal{O}_K$ can be encoded by using

$$\begin{array}{rccc} & \mathcal{O}_K & \longrightarrow & \mathcal{O}_K/\mathfrak{p}_1 \times \cdots \times \mathcal{O}_K/\mathfrak{p}_n \\ c: & m & \longmapsto & (m \bmod \mathfrak{p}_1, \cdots, m \bmod \mathfrak{p}_n). \end{array}$$

The construction of good codes on number fields have been independently studied by Lenstra [Len86] and Guruswami [Gur03]. They provided indications on how to chose number fields having good properties for the underlying codes. In particular, Guruswami [Gur03] showed the existence of asymptotically good number field codes, that is a family $\mathcal{C}_i$ of $[n_i, k_i, d_i]_q$ codes of increasing block length with

$$\liminf \frac{k_i}{n_i} > 0 \text{ and } \liminf \frac{d_i}{n_i} > 0.$$

Neither of them could provide a decoding algorithm. In the concluding remarks of [Gur03], Guruswami identifies the application of the decoding paradigm of [Gur04, GS98, GSS00] to number field codes as an open problem.

Contribution: The main contribution of this paper is to provide the first algorithm for decoding number field codes. We first show that a direct adaptation of an analogue of Coppersmith's theorem due to Cohn and Heninger [CH11] allows to follow the approach of Boneh [Bon00] which does not allow to reach the Johnson bound. Then we adapt the decoding paradigm of [Gur04, Chap. 7] to number field codes, by using methods for manipulating modules over the ring of integers of a number field recently described in [BF12] to achieve the Johnson bound.

Throughout this paper, we denote by K a number field of degree d, of discriminant Δ and of ring of integers $\mathcal{O}_K$. The prime ideals $(\mathfrak{p}_i)_{i \leq n}$ satisfy $\mathcal{N}(\mathfrak{p}_1) < \mathcal{N}(\mathfrak{p}_2) < \cdots < \mathcal{N}(\mathfrak{p}_n)$, and we define $N := \prod_{i \leq n} \mathcal{N}(\mathfrak{p}_i)$ and $B := \prod_{i \leq k} \mathcal{N}(\mathfrak{p}_i)^{1/d}$ for integers k, n such that $0 < k < n$. Before describing our algorithm in more details in the following sections, let us state the main result of the paper.

Theorem 5. *Let $\varepsilon > 0$, and a message $m \in \mathcal{O}_K$ satisfying $\|m\| \leq B$, then there is an algorithm that returns all the messages $m' \in \mathcal{O}_K$ such that $\|m'\| \leq B$ and that $c(m)$ and $c(m')$ have mutual agreement t satisfying*

$$t \geq \sqrt{k(n+\varepsilon)}.$$

This algorithm is polynomial in d , $\log(N)$, $1/\varepsilon$ and $\log|\Delta|$.

6.2 Generalities on number fields

Let K be a number field of degree d. It has $r_1 \leq d$ real embeddings $(\theta_i)_{i \leq r_1}$ and $2r_2$ complex embeddings $(\theta_i)_{r_1 < i \leq r_1 + 2r_2}$ (coming as r_2 pairs of conjugates). The field K is

isomorphic to $\mathcal{O}_K \otimes \mathbb{Q}$ where $\mathcal{O}_K$ denotes the ring of integers of K. We can embed K in

$$K_{\mathbb{R}} := K \otimes \mathbb{R} \simeq \mathbb{R}^{r_1} \times \mathcal{C}^{r_2},$$

and extend the θ_i's to $K_{\mathbb{R}}$. Let T_2 be the Hermitian form on $K_{\mathbb{R}}$ defined by

$$T_2(x, x') := \sum_i \theta_i(x)\overline{\theta_i}(x'),$$

and let $\|x\| := \sqrt{T_2(x, x)}$ be the corresponding L_2-norm. Let $(\alpha_i)_{i \leq d}$ be such that $\mathcal{O}_K = \oplus_i \mathbb{Z}\alpha_i$, then the discriminant of K is given by $\Delta = \det^2(T_2(\alpha_i, \alpha_j))$. The norm of an element $x \in K$ is defined by $\mathcal{N}(x) = \prod_i |\theta_i(x)|$.

We encode our messages with prime ideals of $\mathcal{O}_K$. However, for decoding, we need a more general notion of ideal, namely the fractional ideals of $\mathcal{O}_K$. A subset $\mathfrak{a} \subseteq K$ is said to be a fractional ideal if $\exists r \in \mathbb{Z}, r\mathfrak{a} \subseteq \mathcal{O}_K$. When a fractional ideal is contained in $\mathcal{O}_K$, we refer to it as an integral ideal.two fractional ideals of $\mathcal{O}_K$ is given by

$$\mathfrak{ab} = \{a_1b_1 + \cdots + a_lb_l \mid l \in \mathbb{N}, a_1, \cdots a_l \in \mathfrak{a}, b_1, \cdots b_l \in \mathfrak{b}\}$$

$$\mathfrak{a} + \mathfrak{b} = \{a + b \mid a \in \mathfrak{a}, b \in \mathfrak{b}\}.$$

Any non zero fractional ideal $\mathfrak{a}$ of $\mathcal{O}_K$ is invertible, that is there exists $\mathfrak{a}^{-1} := \{x \in K \mid x\mathfrak{a} \subseteq \mathcal{O}_K\}$ such that $\mathfrak{a}\mathfrak{a}^{-1} = \mathcal{O}_K$. The norm of integral ideals is given by $\mathcal{N}(I) := [\mathcal{O}_K : I]$, which extends to fractional ideals by $\mathcal{N}(I/J) := \mathcal{N}(I)/\mathcal{N}(J)$. The norm of a principal ideal agrees with the norm of its generator $\mathcal{N}(x\mathcal{O}_K) = |\mathcal{N}(x)|$.

In the following, we will study finitely generated sub $\mathcal{O}_K$-module of $\mathcal{O}_K[y]$. Let $M \subseteq K^l$ be a finitely generated $\mathcal{O}_K$-module. As in [Coh91, Chap. 1], we say that $[(a_i), (\mathfrak{a}_i)]_{i \leq n}$, where $a_i \in K$ and $\mathfrak{a}_i$ is a fractional ideal of K, is a pseudo-basis for M if $M = \mathfrak{a}_1 a_1 \oplus \cdots \oplus \mathfrak{a}_n a_n$. We also call a pseudo-matrix representing M the matrix of the coefficients of the $(a_i)_{i \leq n}$ along with the ideals $\mathfrak{a}_i$. The algorithm [BF12, Alg.4] returns a pseudo-matrix representing M where the matrix of the $(a_i)_{i \leq n}$ has a triangular shape in polynomial time.

6.3 Decoding with Coppersmith's theorem

An analogue of Coppersmith's theorem was described by Cohn and Heninger in [CH11]. It was used to provide an elegant way of decoding Reed-Solomon codes, and the possibility to use it for breaking lattice- based cryptosystems in $\mathcal{O}_K$ modules was considered, although they concluded that it would not improve the state-of-the-art algorithms.

Theorem 6 (Cohn-Heninger). *Let $f \in \mathcal{O}_K[X]$ be a monic polynomial of degree l, $0 < \beta \leq 1$, $\lambda_1, \cdots, \lambda_d > 0$ and $I \subsetneq \mathcal{O}_K$ an ideal. We can find in polynomial time all the $\omega \in \mathcal{O}_K$ such that $|\omega|_i := |\sigma_i(\omega)| \leq \lambda_i$ and*

$$\mathcal{N}(\gcd(f(\omega)\mathcal{O}_K, I) > \mathcal{N}(I)^{\beta},$$

provided that the λ_i satisfy $\prod_i \lambda_i < (2 + o(1))^{-d^2/2}\mathcal{N}(I)^{\beta^2/l}$.

Although not mentioned in [CH11], a straightforward adaptation of Theorem 6 with $\beta := \sqrt{\frac{\sum_{i\leq k}\log\mathcal{N}(\mathfrak{p}_i)}{\sum_{i\leq n}\log\mathcal{N}(\mathfrak{p}_i)}}$ where $0 < k < n$, $I := \prod_{i\leq n}\mathfrak{p}_i$ and $\forall i, \lambda_i := \prod_{i\leq k}\mathcal{N}(\mathfrak{p}_i)^{1/d}$ provides a polynomial time algorithm for decoding number field codes.

Theorem 7. *Let $(r_1,\cdots,r_n)\in\mathcal{O}_K^n$ and $m\in\mathcal{O}_K$ satisfying $\forall i,\ m = r_i \bmod \mathfrak{p}_i$, then Theorem 6 applied to $f(\omega) := \omega - m$ allows to return in polynomial time a list of $m'\in\mathcal{O}_K$ with $\mathcal{N}(m')\leq\prod_{i\leq k}\mathcal{N}(\mathfrak{p}_i)$ that differ from m in at most e places where*

$$e < n - \sqrt{kn\frac{\log\mathcal{N}(\mathfrak{p}_n)}{\log\mathcal{N}(\mathfrak{p}_1)}}.$$

In the rest of the paper, we present a method based on Guruswami's general framework for residue codes [Gur04] that allows us to get rid in the dependency in $\frac{\log\mathcal{N}(\mathfrak{p}_n)}{\log\mathcal{N}(\mathfrak{p}_1)}$ in the decoding bound thus reaching the Johnson bound.

6.4 Johnson-type bound for number fields codes

A Johnson-type bound is a positive number J depending on the distance, the blocklength and the cardinalities of the alphabets constituting the code. It guaranties that a "small" number of codewords are in any sphere of radius J. By "small" number, we mean a number of codewords which is linear in the code blocklength and the dimension of the code. In our case, the Johnson-type bound for number fields codes depends only on the code blocklength and its minimal distance, and "small" means polynomial in $\sum_{i=1}^{n}\log\mathcal{N}(\mathfrak{p}_i)$.

The Johnson-type bound of [Gur04, Section 7.6.1] remains valid for number field codes. For any prime ideal $\mathfrak{p}\subset\mathcal{O}_K$, the quotient $\mathcal{O}_K/\mathfrak{p}$ is a finite field. Thus the i'th symbol of a codeword comes from an alphabet of size $\mathcal{N}(\mathfrak{p}_i) = |\mathcal{O}_K/\mathfrak{p}_i|$ and [Gur04, Th. 7.10] can be applied. Let t be the least positive integer such that $\prod_{i=1}^{t}\mathcal{N}(\mathfrak{p}_i) > \left(\frac{2B}{d}\right)^d$, where $d = [K:\mathbb{Q}]$ and let $T = \prod_{i=1}^{t}\mathcal{N}(\mathfrak{p}_i)$. Then, by [Gur03, Lem. 12], the minimal hamming distance of the number fields code is at least $n-t+1$. Using [Gur04, Th. 7.10], we can show that for a given message and $\varepsilon > 0$, only a "small" number of codewords satisfy

$$\sum_{i=1}^{n} a_i > \sqrt{(t+\varepsilon)n}, \tag{6.1}$$

where $a_i = 1$ if the codeword and the message agree at the i-th position, $a_i = 0$ otherwise. Thus, if our list decoding algorithm returns all the codewords having at most $n - \sqrt{(t+\varepsilon)n}$ errors then this number is guaranteed to be "small". Therefore, the Johnson bound appears to be a good objective for our algorithm. Note that we would derive a different bound by using weighted distances. In particular, by using the log-weighted hamming distance i.e. $d(x,y) = \sum_{i:x\neq y \bmod \mathfrak{p}_i}\log\mathcal{N}(\mathfrak{p}_i)$, the condition would be $\sum_{i=1}^{n} a_i\log\mathcal{N}(\mathfrak{p}_i) > \sqrt{(\log T+\varepsilon)\log N}$.

6.5 General description of the algorithm

In this section, we give a high-level description of our decoding algorithm. We follow the approach of the general framework described in [Gur04], making the arrangements required in our context. Our code is the set of $m \in \mathcal{O}_K$ such that $\|m\| \leq B$ where $B = \prod_{i \leq k} \mathcal{N}(\mathfrak{p}_i)^{1/d}$. We also define $N := \prod_{i \leq n} \mathcal{N}(\mathfrak{p}_i)$. A codeword m is encoded via

$$\begin{array}{ccc} \mathcal{O}_K & \longrightarrow & \mathcal{O}_K/\mathfrak{p}_1 \times \cdots \times \mathcal{O}_K/\mathfrak{p}_n \\ m & \longmapsto & (m \bmod \mathfrak{p}_1, \cdots, m \bmod \mathfrak{p}_n). \end{array}$$

Let $z_1, \cdots, z_n$ be non-negative real numbers, and let Z be a parameter. In this section, as well as in Section 6.6 and 6.7, we assume that the z_i are integers. We assume that we received a vector $(r_1, \cdots, r_n) \in \prod_i \mathcal{O}_K/\mathfrak{p}_i$. We wish to retrieve all the codewords m such that $\sum_i a_i z_i > Z$ where $a_i = 1$ if $m \bmod \mathfrak{p}_i = r_i$ and 0 otherwise (we say that m and $(r_i)_{i \leq n}$ have weighted agreement Z).

We find the codewords m with desired weighted agreement by computing roots of a polynomial $c \in \mathcal{O}_K[y]$ that satisfies

$$\|m\| \leq B \Longrightarrow \|c(m)\| < F, \tag{6.2}$$

for an appropriate bound F. We choose the polynomial c satisfying 6.2 in the ideal $\prod_{i \leq n} J_i^{z_i} \subseteq \mathcal{O}_K[y]$ where

$$J_i = \{a(y)(y - r_i) + p \cdot b(y) \mid a, b \in \mathcal{O}_K[y] \text{ and } p \in \mathfrak{p}_i\}.$$

With such a choice of a polynomial, we necessarily have $c(m) \in \prod_i \mathfrak{p}_i^{z_i a_i}$, where $a_i = 1$ if $c(m) \bmod \mathfrak{p}_i = r_i$, 0 otherwise. In particular, if $c(m) \neq 0$ then $\mathcal{N}(c(m)) \geq \prod_i \mathcal{N}(\mathfrak{p}_i)^{z_i a_i}$. In addition, we know, from the inequality between arithmetic and geometric mean, that $\|c(m)\| \geq \sqrt{d}\mathcal{N}(c(m))^{1/d}$. We thus know that if the weighted agreement satisfies

$$\sum_{i \leq n} a_i z_i \log \mathcal{N}(\mathfrak{p}_i) > -\frac{d}{2} \log(d) + d \log(F), \tag{6.3}$$

which in turns implies $\sqrt{d}\,(\prod_i \mathcal{N}(\mathfrak{p}_i)^{z_i a_i})^{1/d} > F$, then $c(m)$ has to be zero, since otherwise it would contradict 6.2.

Algorithm 29 Decoding algorithm

Input: $\mathcal{O}_K$, $z_1, \cdots, z_n$, B, Z, $r_1, \cdots, r_n \in \prod_i \mathcal{O}_K/\mathfrak{p}_i$.
Output: All m such that $\sum_i a_i z_i > Z$.

1: Compute l and F.
2: Find $c \in \prod_{i \leq n} J_i^{z_i} \subseteq \mathcal{O}_K[y]$ of degree at most l such that $\|m\| \leq B \Longrightarrow \|c(m)\| < F$.
3: Find all roots of c and report those roots ξ such that $\|\xi\| \leq B$ and $\sum_i a_i z_i > Z$.

6.6 Existence of the decoding polynomial

In this section, given weights $(z_i)_{i\leq n}$, we prove the existence of a polynomial $c \in \prod_i J_i^{z_i}$ and a constant $F > 0$ such that for all $\|m\| \leq B$, $m \in \mathcal{O}_K$, we have $\|c(m)\| \leq F$. This proof is not constructive. The actual computation of this polynomial will be described in Section 6.7. We first need to estimate the number of elements of $\mathcal{O}_K$ bounded by a given size.

Lemma 249. *Let $F' > 0$ and $0 < \gamma < 1$, then the number of $x \in \mathcal{O}_K$ such that $\|x\| \leq F'$ is at least*

$$\left\lfloor \frac{\pi^{d/2} F'^d}{2^{r_1+r_2-1+\gamma}\sqrt{|\Delta|}\Gamma(d/2)} \right\rfloor.$$

Proof. As in [Neu99, Chap. 5], we use the standard results of Minkowski theory for our purposes. More precisely, there is an isomorphism $f : K_{\mathbb{R}} \longrightarrow \mathbb{R}^{r_1+2r_2}$ and a scalar product $(x, y) := \sum_{i\leq r_1} x_i y_i + \sum_{r_1<i\leq r_1+2r_2} 2x_i y_i$ on $\mathbb{R}^{r_1+2r_2}$ transferring the canonical measure from $K_{\mathbb{R}}$ to $\mathbb{R}^{r_1+2r_2}$. Let $\lambda = f(\mathcal{O}_K)$, $X := \{x \in K_{\mathbb{R}} \mid \|x\| \leq F'\}$, and $m \in \mathbb{N}$. We know from Minkowski's lattice point theorem that if $\mathrm{Vol}(X) > m2^d \det(\lambda)$, then $\#(f(x) \cap \lambda) \geq m$. As $\mathrm{Vol}(X) = 2^{r_2}\left(2\pi^{d/2}F'^d/\Gamma(d/2)\right)$ and $\det(\lambda) = \sqrt{|\Delta|}$, we have the desired result. □

Then, we must derive from Lemma 249 an analogue of [Gur04, Lemma 7.6] in our context. This lemma allows us to estimate the number of polynomials of degree l satisfying 6.2. To simplify the expressions, we use the following notation in the rest of the paper

$$\alpha_{d,\Delta,\gamma} := \frac{\pi^{d/2}}{2^{r_1+r_2-1+\gamma}\sqrt{|\Delta|}\Gamma(d/2)}.$$

Lemma 250. *For positive integers B, F', the number of polynomials $c \in \mathcal{O}_K[y]$ of degree at most l satisfying 6.2 is at least*

$$\left(\alpha_{d,\Delta,\gamma}\left(\frac{F'}{(l+1)B^{l/2}}\right)^d\right)^{l+1}.$$

Proof. Let $c(y) = c_0 + c_1 y + \cdots + c_l y^l$. We want the c_i's to satisfy $\|c_i m^i\| < F'/(l+1)$ whenever $\|m\| \leq B$. This is the case when $\|c_i\| < F'/(B^i(l+1))$. By Lemma 249, there are at least $\alpha_{d,\Delta,\gamma}\left(F'/((l+1)B^i)\right)^d$ possibilities for c_i. Therefore, the number of polynomials c satisfying 6.2 is at least

$$(\alpha_{d,\Delta,\gamma})^{l+1}\left(\left(\frac{F'}{l+1}\right)^{l+1}\prod_{i=0}^{l} B^{-i}\right)^d,$$

which finishes the proof. □

Now that we know how to estimate the number of $c \in \mathcal{O}_K[y]$ of degree at most l satisfying 6.2, we need to find a lower bound on F to ensure that we can find such a polynomial in $\prod_i J_i^{z_i}$. The following lemma is an equivalent of [Gur04, Lemma 7.7].

Lemma 251. *Let l, B, F be positive integers, there exists $c \in \prod_i J_i^{z_i}$ satisfying 6.2 provided that*

$$F > 2(l+1)B^{l/2}\frac{1}{(\alpha_{d,\Delta,\gamma})^{1/d}}\left(\prod_i \mathcal{N}(\mathfrak{p}_i)^{\binom{z_i+1}{2}}\right)^{\frac{1}{d(l+1)}}. \tag{6.4}$$

Proof. Let us apply Lemma 250 to $F' = F/2$. There are at least

$$\left(\alpha_{d,\Delta,\gamma}\left(\frac{F/2}{(l+1)B^{l/2}}\right)^d\right)^{l+1}$$

polynomial $c \in \mathcal{O}_K[y]$ satisfying $\|m\| \leq B \Rightarrow \|c(m)\| < F/2$. In addition, we know from [Gur04, Corollary 7.5] that $\prod_i |\mathcal{N}(\mathfrak{p}_i)|^{\binom{z_i+1}{2}} \geq |\mathcal{O}_K[y]/\prod_i J_i^{z_i}|$, which implies that if 6.4 is satisfied, then necessarily

$$\left(\alpha_{d,\Delta,\gamma}\left(\frac{F/2}{(l+1)B^{l/2}}\right)^d\right)^{l+1} > \left|\mathcal{O}_K[y]/\prod_i J_i^{z_i}\right|.$$

This means that there are at least two distinct polynomials $c_1, c_2 \in \mathcal{O}_K[y]$ of degree at most l such that $(c_1 - c_2) \in \prod_i J_i^{z_i}$ and $\|c_1(m)\|, \|c_2(m)\| < F/2$ whenever $\|m\| \leq B$. The choice of $c := c_1 - c_2$ finishes the proof. □

6.7 Computation of the decoding polynomial

Let $l > 0$ be an integer to be determined later. To compute $c \in \prod_i J_i^{z_i}$ of degree at most l satisfying 6.2, we need to find a short pseudo-basis of the sub $\mathcal{O}_K$-module $M \cap \prod_i J_i^{z_i}$ of K^{l+1} where M is the $\mathcal{O}_K$-module of the elements of $\mathcal{O}_K[y]$ of degree at most l embedded in K^{l+1} via $\sum_i c_i y^i \to (c_i)$. We first compute a pseudo-generating set for each $M \cap J_i^{z_i}$, then we compute a pseudo-basis for their intersection, and we finally call the algorithm of [FS10] to produce a short pseudo-basis of $M \cap \prod_i J_i^{z_i}$ from which we derive c.

An algorithm for computing a pseudo-basis of the intersection of two modules given by their pseudo-basis is described by Cohen in [Coh91, 1.5.2]. It relies on the HNF algorithm for $\mathcal{O}_K$-modules. The HNF algorithm presented in [Coh91, 1.4] is not polynomial, but a variant recently presented in [BF12] enjoys this property. We can therefore apply [Coh91, 1.5.2] with the HNF of [BF12] successively for each pseudo-basis of $M \cap J_i^{z_i}$ to produce a pseudo-basis of $M \cap \prod_i J_i^{z_i}$.

Algorithm 30 Computation of the decoding polynomial

Input: $(\mathfrak{p}_i, z_i)_{i\leq n}$, l, N, B, F such that $\exists c \in \prod_i J_i^{z_i}$ of degree at most l satisfying 6.2 for F, and the encoded message $(r_1, \cdots, r_n) \in \prod_i \mathcal{O}_K/\mathfrak{p}_i$.

Output: $c \in \prod_i J_i^{z_i}$ satisfying 6.2 for $F' = 2^{\frac{dl}{2}}\sqrt{l+1}\left(2^{2+d(6+3d)}d^3|\Delta|^{2+\frac{11}{2d}}\right)F$ of degree at most l.

1: **for** $i \leq n$ **do**
2: $\tilde{z_i} \leftarrow \min(z_i, l)$.
3: For $0 \leq j \leq \tilde{z_i}$: $\mathfrak{a}_j^i \leftarrow \mathfrak{p}_i^{z_i - j}$, $a_j^i \leftarrow (y - r_i)^j$.
4: For $1 \leq j \leq l - z_i$: $\mathfrak{a}_j^i \leftarrow \mathcal{O}_K$, $a_j^i \leftarrow y^j(y - r_i)^{z_i}$.
5: Let $\left((a_j^i), (\mathfrak{a}_j^i)_{j\leq l+1}\right)$ be a pseudo matrix for $M \cap J_i^{z_i}$.
6: **end for**
7: Compute a pseudo-basis $[(c_i), (\mathfrak{c}_i)]_{i\leq l+1}$ of $M_1 = M \cap \prod_i J_i^{z_i}$.
8: Deduce a pseudo-basis $[(d_i), (\mathfrak{d}_i)]_{i\leq l+1}$ of the module M_2 given by

$$(v_0, v_1, \cdots, v_l) \in M_1 \Longleftrightarrow (v_0, v_1 \cdot B, \cdots, v_l \cdot (B)^l) \in M_2.$$

9: Let $[(b_i), (\mathfrak{b}_i)]_{i\leq l+1}$ be a short pseudo-basis of M_2 obtained with the reduction algorithm of [FS10].
10: Let x_1, x_2 be a short basis of $\mathfrak{b}_1$ obtained with [FS10, Th. 3].
11: **return** $c \in M_1$ corresponding to $x_1 b_1 \in M_2$.

6.8 Good weight settings

To derive our main result, we need to consider weights $z_i > 0$ in $\mathbb{R}$ rather than $\mathbb{Z}$. Let

$$\beta_{d,\Delta,\gamma} := \frac{d^{3-\frac{d}{2}}2^{3(1+d(2+d))}|\Delta|^{2+\frac{11}{2d}}}{\alpha_{d,\Delta,\gamma}{}^{\frac{1}{d}}},$$

then by combining 6.3, 6.4 and Algorithm 30, we know that given $(r_1, \cdots, r_n) \in \prod_{i\leq n} \mathcal{O}_K/\mathfrak{p}_i$, $l > 0$, $B = \prod_{i\leq k} \mathcal{N}(\mathfrak{p}_i)^{1/d}$ and integer weights $z_i > 0$, Algorithm 30 returns a polynomial c of degree at most l such that all $m \in \mathcal{O}_K$ satisfying $\|m\| \leq B$ and

$$\sum_{i\leq n} a_i z_i \log \mathcal{N}(\mathfrak{p}_i) \geq \frac{l}{2}\log(2^{d^2}B^d) + \frac{3d}{2}\log(l+1) + \frac{1}{l+1}\sum_{i\leq n}\binom{z_i+1}{2}\log \mathcal{N}(\mathfrak{p}_i) + \log \beta_{d,\Delta,\gamma}, \tag{6.5}$$

(where $a_i = 1$ if $m \bmod \mathfrak{p}_i = r_i$, 0 otherwise) are roots of c. In the following, we no longer assume the z_i to be integers. However, we will use our previous results with the integer weights $z_i^* := \lceil Az_i \rceil$ for a sufficiently large integer A to be determined.

Proposition 252. *Let $\varepsilon > 0$, non-negative reals z_i, $B = \prod_{i \le k} \mathcal{N}(\mathfrak{p}_i)^{1/d}$, and an encoded message $(r_1, \cdots, r_n) \in \prod_i \mathcal{O}_K/\mathfrak{p}_i$, then our algorithm finds all the $m \in \mathcal{O}_K$ such that $\|m\| \le B$ and*

$$\sum_{i \le n} a_i z_i \log \mathcal{N}(\mathfrak{p}_i) \ge \sqrt{\log(2^{d^2} B^d) \left(\sum_{i \le n} z_i^2 \log \mathcal{N}(\mathfrak{p}_i) + \varepsilon z_{max}^2 \right)},$$

where $a_i = 1$ if $m \bmod \mathfrak{p}_i = r_i$, 0 otherwise.

Proof. Note that we can assume without loss of generality that $z_{max} = 1$. Let $z_i^* = \lceil A z_i \rceil$ for a sufficiently large integer A, which thus satisfies $A z_i \le z_i^* < A z_i + 1$. The decoding condition 6.5 is met whenever

$$\begin{aligned} \sum_{i \le n} a_i z_i \log \mathcal{N}(\mathfrak{p}_i) \ge{}& \frac{l}{2A} \log(2^{d^2} B^d) + \frac{3d}{2A} \log(l+1) \\ &+ \frac{A}{2(l+1)} \sum_{i \le n} \left(z_i^2 + \frac{3}{A} z_i + \frac{2}{A^2} \right) \log \mathcal{N}(\mathfrak{p}_i) \\ &+ \frac{1}{A} \log \beta_{d,\Delta,\gamma}. \end{aligned} \tag{6.6}$$

Let $Z_i := z_i^2 + \frac{3}{A} z_i + \frac{2}{A^2}$ for $i \le n$ and

$$l := \left\lceil A \sqrt{\frac{\sum_{i \le n} Z_i \log \mathcal{N}(\mathfrak{p}_i)}{\log(2^{d^2} B^d)}} \right\rceil - 1.$$

We assume that $A \ge \log(2^{d^2} B^d)$, which ensures that $l > 0$. For this choice of l, condition 6.6 is satisfied whenever

$$\begin{aligned} \sum_{i \le n} a_i z_i \log \mathcal{N}(\mathfrak{p}_i) \ge{}& \frac{3d}{2A} \log \left(A \sqrt{\frac{\sum_{i \le n} Z_i \log \mathcal{N}(\mathfrak{p}_i)}{\log(2^{d^2} B^d)}} + 1 \right) \\ &+ \sqrt{\log(2^{d^2} B^d) \left(\sum_{i \le n} Z_i \log \mathcal{N}(\mathfrak{p}_i) \right)} \\ &+ \frac{1}{A} \log \beta_{d,\Delta,\gamma}. \end{aligned} \tag{6.7}$$

Assume that $A \ge \frac{10 \log N}{\varepsilon}$ and $A \ge \frac{\log \beta_{d,\Delta,\gamma}}{\log N}$, then for N large enough, the right side

of 6.7 is at most

$$O\left(\frac{\log\log N}{\log N}\right) + \sqrt{\log(2^{d^2}B^d)\left(\sum_{i\leq n} z_i^2 \log\mathcal{N}(\mathfrak{p}_i) + \frac{\varepsilon}{2}\right)}$$
$$\leq \sqrt{\log(2^{d^2}B^d)\left(\sum_{i\leq n} z_i^2 \log\mathcal{N}(\mathfrak{p}_i) + \varepsilon\right)}$$

The degree l of our decoding polynomial c is therefore polynomial in $\log N$, $\frac{1}{\varepsilon}$, d and $\log|\Delta|$. By [Aya10, 2.3], we know that the complexity to find the roots of c is polynomial in d, l and in the logarithm of the height of c, which we already proved to be polynomial in the desired values.

□

Corollary 253. *Let $\varepsilon > 0$, $k < n$ and prime ideals $\mathfrak{p}_1, \cdots \mathfrak{p}_n$ satisfying $\mathcal{N}(\mathfrak{p}_i) < \mathcal{N}(\mathfrak{p}_{i+1})$ and $\log\mathcal{N}(\mathfrak{p}_{k+1}) \geq (k\log\mathcal{N}(\mathfrak{p}_k) + d^2)$, then with the previous notations, our algorithm finds a list of all codewords which agree with a received word in t places provided $t \geq \sqrt{k(n+\varepsilon)}$.*

Proof. The proof is similar to the one of [Gur04, Th. 7.14]. The main difference is that we define $\delta := k - \frac{\log(2^{d^2}B^d)}{\log\mathcal{N}(\mathfrak{p}_{k+1})}$ which satisfies $\delta \geq 0$ since by assumption $\log\mathcal{N}(\mathfrak{p}_{k+1}) \geq (k\log\mathcal{N}(\mathfrak{p}_k) + d^2)$. We apply Proposition 252 with $z_i = 1/\log\mathcal{N}(\mathfrak{p}_i)$ for $i \geq k+1$, $z_i = 1/\log\mathcal{N}(\mathfrak{p}_{k+1})$ for $i \leq k$, and $\varepsilon' = \varepsilon/\log\mathcal{N}(\mathfrak{p}_{k+1})$. It allows us to retrieve the codewords whose number of agreements t is at least

$$\sqrt{\frac{\log(2^{d^2}B^d)}{\log\mathcal{N}(\mathfrak{p}_{k+1})}\left(\frac{\log(B)}{\log\mathcal{N}(\mathfrak{p}_{k+1})} + \sum_{i=k+1}^{n}\frac{\mathcal{N}(\mathfrak{p}_{k+1})}{\log\mathcal{N}(\mathfrak{p}_i)} + \varepsilon'\right)}$$
$$\leq \delta + \sqrt{\frac{\log(2^{d^2}B^d)}{\log\mathcal{N}(\mathfrak{p}_{k+1})}\left(\frac{\log(2^{d^2}B^d)}{\log\mathcal{N}(\mathfrak{p}_{k+1})} + \sum_{i=k+1}^{n}\frac{\mathcal{N}(\mathfrak{p}_{k+1})}{\log\mathcal{N}(\mathfrak{p}_i)} + \varepsilon\right)}.$$

This condition is met whenever $t \geq \delta + \sqrt{(k-\delta)(n-\delta+\varepsilon)}$. From the Cauchy-Schwartz inequality, we notice that

$$\sqrt{k(n+\varepsilon)} \geq \sqrt{(k-\delta)(n-\delta+\varepsilon)},$$

which proves that our decoding algorithm works when $t \geq \sqrt{k(n+\varepsilon)}$. □

6.9 Conclusion

We presented the first method for list decoding number field codes. A straightforward application of Theorem 6 allows to derive a decoding algorithm in polynomial time.

However, we cannot achieve the Johnson bound with this method. To solve this problem, we described an analogue of the CRT list decoding algorithm for codes based on number fields. This is the first algorithm allowing list decoding of number field codes up to the Johnson bound. We followed the approach of [Gur04, Ch. 7] that provides a general frameworks for list decoding of algebraic codes, along with its application to CRT codes. The modifications to make this strategy efficient in the context of number fields are substantial. We needed to refer to the theory of modules over a Dedekind domain, and carefully analyze the process of intersecting them, as well as finding short elements. We proved that our algorithm is polynomial in the size of the input, that is in d, $\log(N)$, $\log|\Delta|$ and $\frac{1}{\varepsilon}$.

Acknowledgment

The first author would like to thank Guillaume Hanrot for his helpful comments on the approach based on Coppersmith's theorem. We also thank an anonymous referee for helpful comments on this paper.

Part IV

Implementation

In this part I present the implementation in C/C++ of some algorithms presented in the first two parts of the PhD thesis. First in Chapter 7, I present the C++ computer algebra system `Mathemagix` where I have implemented finite fields arithmetic with a special emphasis for finite fields of characteristic 2 in the `finitefiedz` package. I also present the implementation if the arithmetic of Galois rings and the root finding of univariate and bivariate polynomials over Galois rings and truncated power series rings in the `quintix` package. Then in Chapter 8, I present the implementation in C of list decoding algorithms in an independant library called `decoding`. This library allows one to have list decoding algorithms without having to install a while computer algebra system. To my knowledge there was no open source implementation available for list decoding generalized Reed-Solomon codes thus it was necessary to have a working implementation.

Chapter 7

Implementation within Mathemagix

7.1 Introduction

I began to contribute to the Mathemagix computer algebra system in the end of 2009. Mathemagix provides a new high level language, which is imperative, strongly typed, with polymorphism and parametrized types. Mathemagix can be used as an "extension language", i.e. easy to embed into other applications and to extend with existing libraries written in other languages like C or C++. An interesting feature is that this extension mechanism supports template types.

Standard libraries are available for algebraic computation (large numbers, polynomials, power series, matrices, etc. based on FFT and other fast algorithms) for exact and approximate computation. This should make Mathemagix particularly suitable as a bridge between symbolic computation and numerical analysis. These packages written in C++ are connected to the interpreter (and later to the compiler), but can also be used independently as standalone libraries. Separate documentation for each of the packages is also available.

I wrote, with the help of Grégoire Lecerf, three packages. I first wrote the mgf2x package. It is a wrapper for the gf2x library [BGTZ08, BGTZ09] which is a highly optimized C library specialized in univariate polynomial multiplications over $\mathbb{F}_2$. Then I wrote the finitefieldz package which provides finite fields, the main fields used in coding theory. Finally I wrote the quintix package which provides Galois rings arithmetic and the univariate root finding algorithms of Berthomieu, Lecerf and Quintin [BLQ11].

All package are written in C++ and uses C++ templates to for efficiency reasons. They allow functions and classes to operate with generic types and avoid typing functions and classes several times for each type you want to use. Moreover the compiler usually knows what types are used within templates which allows it to do a lot of optimizations.

Mathemagix can be downloaded from http://www.mathemagix.org/www/main/index.en.html. The latest *development* version can be obtained via svn.

```
svn checkout svn://scm.gforge.inria.fr/svnroot/mmx/
```

7.2 Overview of the C++ side of Mathemagix

7.2.1 The directory tree of Mathemagix

The directory tree of Mathemagix is quite simple. Basically every subdirectory of the root directory of Mathemagix represents a package. A package is nothing than a library. It can be a dynamic library (.so files), a static library (.a files) or both. We list a few available packages.

- basix contains, among others, the implementation of vectors, lists and tables. It is the core library of Mathemagix upon which all other libraries rely.
- numerix contains, among others, the implementation of multiprecision integers, rational, complex numbers, intervals and balls.
- algebramix contains, among others, the implementation of univariate polynomials, power series and matrices.
- finitefieldz contains the implementation of finite fields.
- mpari contains a wrapper to the PARI library [PAR11].
- quintix is my package which contains the algorithms that I implemented in Mathemagix. Usually the packages named after a person, like gregorix or jorix, contain versions in development of algorithms or features which, when considered stable, are moved to other packages.

Packages aim to be as independent as possible from each other. But some packages depend on others. For example the finitefieldz package depends on algebramix and numerix to provide large prime fields and extensions of finite fields. These two dependencies allow the finitefieldz package to benefit from all the optimizations and fast algorithms implemented in numerix and algebramix. The build system of Mathemagix takes care of the dependencies. For example if you type

```
./configure --enable-finitefieldz
```

you will see that the algebramix and numerix packages are automatically selected by the build system as well as the core package basix.

```
Packages:
[ ] automagix
[*] basix
[ ] borderbasix
[ ] mcoq
[ ] mmancient
[ ] mmcompiler
[ ] mmdoc
[ ] mmxtools
```

```
[*] numerix
[*] algebramix
[ ] analyziz
[*] finitefieldz
[ ] graphix
[ ] holonomix
[ ] lattiz
[ ] linalg
[ ] mgf2x
[ ] mmaple
[ ] mpari
[ ] multimix
[ ] continewz
[ ] factorix
[ ] gregorix
[ ] mfgb
[ ] quintix
[ ] realroot
[ ] mmps
[ ] newmac
[ ] polytopix
[ ] shape
[ ] symbolix
[ ] asymptotix
[ ] columbus
[ ] jorix
[ ] mmxlight
configure: creating ./config.status
config.status: creating Makefile
config.status: creating mmxlight/src/mmxlight_evaluator.cpp
```

Then you simply have to type `make` to build only the packages (or libraries) you want to use.

When opening a package directory, let say, `finitefieldz`, there are many subdirectories, common to almost all packages:

- `doc/` contains the documentation files for the package.
- `specif/` contains the specifications of the package like its name, version, dependencies. As an example the `specif/finitefieldz-autotools.mmx` file contains the following which is self-understandable:

```
finitefieldz: Package := package ("finitefieldz", "0.1");
finitefieldz.dependencies := [ "algebramix" ];
finitefieldz.externals := [ "mpfq" ];
```

```
finitefieldz.documentation := false;
finitefieldz.automatic := [
      "configure.ac",
      "Makefile.am",
      "build/Makefile.am",
      "script/finitefieldz-config.in",
      "man/man1/finitefieldz-config.1.in" ];
finitefieldz.description :=
  /" Finite fields. "/;

create_package (finitefieldz);
```

- `test/` contains `C++` source files to test the algorithms and functionalities provided by the package.
- `bench/` contains `C++` source files to bench the algorithms and functionalities provided by the package.
- `include/finitefieldz/` contains `C++` header files for the algorithms and features provided by the package.
- `src/finitefieldz/` also contains `C++` source files for the algorithms and features provided by the package. Almost all the `C++` code uses many template classes, structures and functions which must go into `C++` header files with their bodies. This is a consequence of the design of `C++`. Therefore almost all the code of the libraries (or packages) are in the `include/finitefieldz` folder.

7.2.2 `C++` classes and variants

We first present what is the general philosophy behind the `C++` libraries of `Mathemagix`. `C++` templates are heavily used. They help write generic code and, at the same time, help the compiler to perform some optimizations. Note that the listings of `C++` code we present in this section are *not* part of `Mathemagix`. They are to be considered as examples to understand how `Mathemagix` works.

Let say we want to implement vectors as n-tuples over any kind of mathematical object that has an addition. Each object is implemented in a separate `C++` header file (.hpp). Suppose that we implement our tuples in the `myvector.hpp` header file within the `algebramix` package.

```
#include <basix/basix.hpp>

template< typename R, typename V >
class myvector {
  ...
};
```

The first template parameter of the template `myvector` class is a `C++` class known as `R` within the `myvector` class. It represents the mathematical object with which the vectors are made of. For example it can be a field, a ring or any implemented class like the following one implemented in the `myint.hpp` header file in the `algebramix` package.

```
#include <basix/basix.hpp>

class myint {
  int n;

  /* a constructor that takes an C++ int to
     set the value of our new myint object. */
  myint (int n) {
    this->n= n;
  }

  myint operator+ (myint& a,myint& b) {
    return myint (a.n + b.n);
  }
};
```

The second parameter of the template `myvector` class is also a `C++` class which will represent the *variant* used by the functions and algorithms manipulating `myvector`-type objects. For example, the `V` parameter can be used to specify that the user wants to use SSE instructions to add two vectors or loop unrolling. Usually default values for template parameters indicating variants are provided. Therefore the user does not need to know all the available variants and just use the default ones.

```
#include <basix/basix.hpp>

struct myvector_default;

template< typename R, typename V= myvector_default >
class myvector {
  ...
};
```

Now new vectors can be created and manipulated. If the user wants to use vectors over "small integers" represented by the `myint C++` class, he will have to type:

```
#include <algebramix/myvector.hpp>
#include <algebramix/myint.hpp>

int main (void) {
  myvector< myint > v, w;
  return 0;
```

```
}
```

The Mathemagix C++ packages come with a lot of types representing various mathematical objects. We list a few of them here.

- integer for multiprecision integers. (numerix/include/numerix/integer.hpp)
- rational for multiprecision rational numbers. (numerix/include/numerix/rational.hpp)
- complex for complex numbers. (numerix/include/numerix/complex.hpp)
- polynomial for univariate polynomials. (algebramix/include/algebramix/polynomial.hpp)

Of course they can be used to make vectors with our myvector class. For example over the rational numbers it suffices to type:

```
#include <algebramix/myvector.hpp>
#include <numerix/rational.hpp>

int main(void) {
  myvector< rational > v;
  return 0;
}
```

The polynomial class is also template and takes two arguments.

```
template< typename C,
          typename V=typename Polynomial_variant(C) >
class polynomial {
  ...
};
```

Therefore, we can use it with our myvector class to make vectors over polynomials over myint numbers.

```
#include <algebramix/myvector.hpp>
#include <algebramix/myint.hpp>
#include <algebramix/polynomial.hpp>

int main(void) {
  myvector< polynomial< myint > > v, w;
  return 0;
}
```

We do not specify any variant and let Mathemagix use the default ones for our class and the polynomial class.

We now give more details about the implementation of variants. We continue with our example and give pieces of code to help fix the ideas. We begin with the `implementation` structure which will help create the variants. Let's say say we propose for the implementation of the component wise addition of two `myvector` classes three algorithms. The first will be the default variant corresponding to a naive `for` loop. The second variant will use the SSE instructions while the third variant will use unrolled loops. All these variants can be written in the `myvector.hpp` file.

```
template<typename F, typename V, typename W=V>
struct implementation;

/* F parameter: variant for which type of operations?
                GCD, multiplications, ...
                In our case: component wise operations.
                Thus we call it linear for linear operations
                in the memory. */
struct myvector_linear;

/* V, W parameters: list of proposed variants */
struct myvector_sse;      // use of the SSE instructions
struct myvector_naive;    /* naive (and default) variant
                             which will be a simple for
                             loop */
struct myvector_unrolled; /* unrolled loop variant */
```

Suppose that we have a constructor for our `myvector` class that takes an `int sz` and create a vector of size `sz` whose elements are accessible with the "`[]`" operator.

```
#include <basix/basix.hpp>

template< typename R, typename V= myvector_naive >
struct myvector {
  R *tab;   // a pointer to the elements of type R
  int size; // the size of tab

  /* the constructor that allocates memory of
     size (sz * sizeof (R)) */
  myvector (int sz) {
    ...
  }

  // return the i'th element of tab
  myvector operator[] (int i) {
    ...
  }
```

```
}
```

The default variant is the naive one. The implementation of the component wise addition is quite simple with the `implementation` structure.

```
#include <basix/basix.hpp>

template< typename R, typename V= myvector_naive >
struct myvector {
  ...
  myvector operator+ (myvector& a,myvector& b) {
    return implementation< myvector_linear, V >::add (a, b);
  }
  ...
}
```

The code that will really perform the addition must be written within the instantiations of the `implementation` structure for all the proposed variants.

```
template< >
struct implementation< myvector_linear, myvector_naive > {
  myvector add (myvector& a, myvector& b) {
    ASSERT (a.size == b.size, "vectors␣must␣same␣length");
    myvector r= myvector (a.size);
    for (int i= 0; i < a.size; i++) {
      r[i]= a[i] + b[i];
    }
    return r;
  }
};

template< >
struct implementation< myvector_linear, myvector_unrolled > {
  myvector add (myvector& a, myvector& b) {
    ASSERT (a.size == b.size, "vectors␣must␣same␣length");
    myvector r= myvector (a.size);
    int i, n= a.size % 4;
    for (i= 0; i < n; i++) {
      r[i]= a[i] + b[i];
    }
    for ( ; i < a.size; i += 4) {
      r[  i  ]= a[  i  ] + b[  i  ];
      r[i + 1]= a[i + 1] + b[i + 1];
      r[i + 2]= a[i + 2] + b[i + 2];
      r[i + 3]= a[i + 3] + b[i + 3];
```

```
    }
    return r;
  }
};

template< >
struct implementation< myvector_linear, myvector_sse > {
  myvector add (myvector& a, myvector& b) {
    ASSERT (a.size == b.size, "vectors must same length");
    myvector r= myvector (a.size);
    int i, n= a.size % 4;
    for (i= 0; i < n; i++) {
      r[i]= a[i] + b[i];
    }

    // use SSE instructions here
    return r;
  }
};
```

Almost all the C++ classes of Mathemagix do not contain the actual representation of objects.

Listing 7.1: basix/include/basix/vector.hpp

```
template< typename C, typename V >
class vector {
  ...
protected:
  vector_rep< C, V >* rep;
  ...
};
```

The vector class contains a field rep which is a pointer to another class vector_rep. It contains the representation of vectors.

Listing 7.2: basix/include/basix/vector.hpp

```
class vector_rep : public rep_struct, public format< C > {
  ...
private:
  C*   a;           // entries of vector
  nat  n;           // dimension of vector
  nat  l;           // allocated number of entries
  bool scalar_flag; // is the vector a scalar value?
  ...
```

```
};
```

The vector_rep structure inherits of the rep_struct structure which contains all the necessary fields for the implementation of a reference counting garbage collector which is used by the Mathemagix interpreter and compiler.

Listing 7.3: basix/include/basix/basix.hpp

```
struct rep_struct {
  ...
  int ref_count;
  inline rep_struct (): ref_count (1) {}
  virtual inline ~rep_struct () {}
};
```

Mathemagix does not depend on the standard template library (STL) library. The standard input/output streams can be used but it is recommended to use they replacement: mmout for standard output, cin for standard input and cerr for error output.

We finish this section with the nat type. The nat type represents the natural integer type of the processor. For x86 and x86_64 processors it is usually a typedef for unsigned int and is therefore a 32-bits unsigned integer. The nat type is usually used to index C++ arrays, Mathemagix lists and vectors.

7.3 The mgf2x package

The mgf2x package is the first package I wrote at the very beginning of my thesis with the help of Grégoire Lecerf. It is a wrapper to the gf2x library [BGTZ08,BGTZ09], more particularly to the gf2x_mul function. It is implemented as an additional variant for polynomials over integers modulo 2. As explained in Subsection 7.2.2 the implementation structure is appropriately instantiated.

Listing 7.4: mgf2x/include/mgf2x/polynomial_gf2x.hpp

```
#include <gf2x.h>
#include <numerix/modular_int.hpp>
#include <algebramix/polynomial.hpp>
#include <algebramix/polynomial_dicho.hpp>

template<typename V>
struct implementation< polynomial_multiply,
                       V,
                       polynomial_gf2x > :
       public implementation<polynomial_linear,V>
{
  ...
public:
```

```
  template< typename I, typename MoV, typename MaV >
  static inline void
  mul (modular< modulus< I, MoV >, MaV >* dest,
       const modular< modulus< I, MoV >, MaV >* s1,
       const modular< modulus< I, MoV >, MaV >* s2,
       nat n1, nat n2)
  {
    // input conversions
    gf2x_mul (d, t1, m1, t2, m2);
    // output conversions
  }
  ...
};
```

The implemented variant is called `polynomial_gf2x` and can be used whenever polynomial over $\mathbb{F}_2$ are needed provided that the `gf2x` library is present on the system.

7.4 The `finitefieldz` package

The `finitefieldz` package provides the arithmetic for finite fields. I wrote this package with the help of Grégoire Lecerf during the first year of my PhD thesis. I wrote finite fields arithmetic using univariate polynomials and multiprecision integers. The user can then use all finite field of any characteristic and can build towers of finite fields. I also included and corrected some bugs in the functions, written by Lecerf, which implement univariate polynomials root finding over finite fields.

The `finitefieldz` package uses the `algebramix` and `numerix` packages which were mainly written by Lecerf and van der Hoeven.

7.4.1 Prime fields

We begin with the construction of prime fields. Choose a prime integer and denote by `p` the corresponding variable in `C++` of type `I`. We do not give `p` an explicit type for now. First we need to build a *modulus* with the `modulus` class, then we can build any elements of $\mathbb{Z}/\mathtt{p}\mathbb{Z}$ using the `modular` class.

```
#include <numerix/modular.hpp>

int main(void) {
  modulus< I > mod= modulus< I > (p);
  modular< modulus >::set_modulus (mod);
  modular< modulus > a, b;
  I another_integer= 12345;
  modular< modulus > c= modular< modulus > (another_integer);
  return 0;
```

```
}
```

We often call `p` or the integer it represents the modulus. As usual `Mathemagix` provides default variants for `modulus` and `modular`. The `modular` class has a local and global variant. The default variant is the global one. All element of type `modular< modulus < I > >` or `modular< modulus< I >, modular_global >` share the same modulus and therefore live in the same prime field. Each element of type `modular< modulus< I >, modular_local >` has its own modulus and therefore lives in its own finite field. If two such elements have the same modulus they live in the same prime field. Suppose that `q` is another variable of type `I` representing a prime.

```
#include <numerix/modular.hpp>

int main(void) {
  modulus< I > modp= modulus< I > (p);
  modulus< I > modq= modulus< I > (q);
  modular< modulus, modular_local > a, b;
  set_modulus (a, p);
  set_modulus (b, q);
  I c= 12345;
  a= modular< modulus< I >, modular_local > (c);
  b= modular< modulus< I >, modular_local > (c);
  mmout << ( a == b ) << "\n";
  return 0;
}
```

The `a` and `b` variables have the same type but live in two different finite fields. We have

"`a` $\in \mathbb{Z}/$`p`$\mathbb{Z}$ and `b` $\in \mathbb{Z}/$`q`$\mathbb{Z}$".

The `mmout` variable will output `0` unless `p == q`. It is recommended to use the non-static version of the `set_modulus` function as it works also with the global variant.

There exists a third variant for `modular`. When the modulus is known at compile time and hold in a machine word (`char`, `short`, `int`, `long` or `long long`) the compiler can do optimizations, for example, in the reduction modulo functions. In this situation one can use the `modular_fixed` variant.

```
#include <numerix/modular.hpp>
#include <numerix/modular_int.hpp>

int main(void) {
  unsigned char p= 5;
  modulus< unsigned char > mod= modulus< unsigned char > (p);
  modular< modulus< unsigned char >,
           modular_fixed< unsigned char, 5 > > a, b;
  return 0;
```

```
}
```

The `modular_fixed` structure is template and takes two arguments. The first argument indicates the type of integer that will be used to represents the elements of $\mathbb{Z}/\mathtt{p}\mathbb{Z}$. The second argument is the number of bits needed to represent `p`.

The `modulus` structure is also template and takes two arguments. The first argument is the integer type that will be used to store the modulus `p`. The second argument is the variant that controls the algorithms related to the arithmetic of $\mathbb{Z}/\mathtt{p}\mathbb{Z}$. We list some of them here.

`template< nat size > struct modulus_int_naive;`
: is the naive variant for machine integers (`int`). The `size` argument is the maximum bit-size allowed for the modulus. This variant is valid only when the modulus is a `C++` integer type.

`template< nat m > struct modulus_int_preinverse;`
: is a variant where a suitable inverse of the modulus is pre-computed. The `m` argument is the maximum bit-size allowed for the modulus. This variant is valid only when the modulus is a `C++` integer type.

`struct modulus_integer_naive;`
: is the variant to be used when the modulus is a multiprecision integer of type `integer`.

7.4.2 Extensions of finite fields

The `C++` class which represents a finite field extension is the template `ffe` class which takes two arguments.

```
#include <basix/list.hpp>
#include <finitefieldz/ffe_naive.hpp>

template< typename M, typename V= typename Ffe_variant(M) >
class ffe {
  ...
};
```

As before the `V` represents the variant for finite fields which has a default value and thus need not be specified. The `M` argument is the base field. In fact the type `ffe` represents an extension of the base field designated by the type `M`. For example, suppose that `M` is any finite field.

```
#include <finitefieldz/ffe.hpp>

int main(void) {
```

```
  ffe< ffe< ffe< M > > > a, b;
  return 0;
}
```

The elements a and b lie in an extension of an extension of an extension of the finite field M. As often, we do not specify any variant and let Mathemagix decides what variant to use.

We do not specify what are degrees of the extensions we build. The degree of an extension is set with the set_extension_degree function which takes an argument of type nat.

```
#include <finitefieldz/ffe.hpp>

int main(void) {
  int d;
  ...
  ffe< M >::set_extension_degree (d);
  return 0;
}
```

The set_extension_degree functions finds an irreducible polynomial of degree d over M and build the extension of M defined by the latter polynomial. If the user wants to use its own irreducible polynomial, he can use the set_defining_polynomial function which takes one argument of type polynomial< M, W >. Suppose that M is $\mathbb{F}_5$ then one can make an extension of degree 3 with $X^3 + X + 1$.

```
#include <finitefieldz/ffe.hpp>
#include <algebramix/polynomial.hpp>

int main(void) {
  polynomial< M > X= polynomial< M > (M (1), 1);
  polynomial< M > phi= (X * X * X) + X + 1;
  ffe< M >::set_defining_polynomial (phi);
  return 0;
}
```

Of course one can obtain all the properties of a finite field. The get_extension_degree functions returns the degree of the extension "[ffe< M >:M]". The get_defining_polynomial returns the irreducible polynomial phi used to build the extension

$$\text{“ffe< M >} = \frac{\text{M}[X]}{(\text{phi}(X))}\text{”}.$$

7.4.3 Variants available for ffe

We now describe how to make two different extensions of M. The set_extension_degree and set_defining_polynomial functions we presented are static functions and thus

apply to each element of type `ffe`. Therefore all theses elements lie in the same extension. To make two different extensions we have to use variants. It is possible to have a general extension degree for all variable of type `ffe< M >` or to define, for each variable of type `ffe< M >`, an extension degree so that two different variable can represent two finite fields elements not lying in the same extension. The so-called *local* variant allows one to have a different extension for each variable of type `ffe< M >`. Suppose that `M` represents $\mathbb{F}_5$.

```
int main(void) {
  polynomial< M > X= polynomial< M > (M (1), 1);
  ffe< M, ffe_naive_local > a, b;
  set_defining_polynomial (a, (X * X) + X  + 1);
  set_defining_polynomial (b, (X * X * X) + X + 1);
  return 0;
}
```

The variable `a` lies in as extension of degree 2 of `M` while `b` lies in as extension of degree 3. The non-static version of the `set_defining_polynomial` is used to set, for each variable, the finite field it lives in. The default variant for the `ffe` class is `ffe_naive` which corresponds to the global variant. Each variable of type `ffe< M, ffe_naive >` or `ffe< M >` shares the same extensions it lives in.

```
int main(void) {
  polynomial< M > X= polynomial< M > (M (1), 1);
  ffe< M, ffe_naive > a;
  ffe< M >            b;
  set_defining_polynomial (a, (X * X) + X  + 1);
  set_defining_polynomial (b, (X * X * X) + X + 1);
  return 0;
}
```

We first set the variable `a` *and* `b` to live in the same extension of degree 2 then of degree 3. In the end both variables lives in the same extension of degree 3 defined by the irreducible polynomial $X^3 + X + 1$ over $\mathbb{F}_5$.

It is recommended to use only the non-static versions of functions as they also work for variables of type `ffe< M, ffe_naive >`. The code produced following this guideline will then be compatible for local and global variant and it will be more generic. We list here some functions to manipulate finite fields extensions.

`polynomial< M > get_defining_polynomial (ffe< M, V >& x);`
returns the defining polynomial of the extension where `x` lives.

`void set_defining_polynomial (ffe< M, V >& x, polynomial< M, W >& p);`
sets the polynomial for the extension where `x` lives. If `x` is of type `ffe< M, ffe_naive >` then all variables of type `ffe< M. ffe_naive >` will live in the same extension.

If however x is of type ffe< M, ffe_naive_local > then only x will live in the extension defined by the polynomial p.

mat get_extension_degree (ffe< M, V >& x);
returns the extension degree "[ffe< M, V > : M]".

void set_extension_degree (ffe< M, V >& x, nat d);
sets the degree of the extension where x lives to d.

integer get_characteristic (ffe< M, V >& x);
returns the characteristic of the field where x lives.

nat get_degree_over_prime_field (ffe< M, V >& x);
returns the degree of the extension where x lives over the prime field. If the prime field is $\mathbb{F}_p$ then it returns "[ffe< M, V > : $\mathbb{F}_p$]" even if M $\neq \mathbb{F}_p$.

The variant for ffe does not only contain whether an element is local or global. It also contains information on the underlying algorithms used for the arithmetic. An element of type ffe< M, V > is represented as a polynomial with coefficients in M. To multiply two elements of type ffe< M, V >, a multiplication of two polynomials of type polynomial< M > is performed. Then the result is reduced modulo the polynomial defining the extension ffe< M, V >. Both the multiplication and the reduction modulo are controlled by the variant V. For more details about the variants see the finitefieldz/include/finitefieldz/ffe_naive.hpp file. It is recommended to let the default variant proposed by Mathemagix as a variant for polynomial dedicated for ffe has been written by Lecerf.

7.4.4 Finite fields of characteristic 2

Finite fields of characteristic 2 can be efficiently implemented. Their elements can be represented in a efficient form within machine words. Their addition becomes then a simple XOR. Finite fields of characteristic 2 can of course be built with the ffe class. However an optimized implementation is proposed with by ffe_2 class. This class is specialized for finite fields of characteristic 2 whose elements can be represented in a machine word.

```
#include <numerix/modular.hpp>
#include <numerix/modular_int.hpp>
#include <finitefieldz/ffe_2.hpp>

int main(void) {
  typedef modular< modulus< unsigned char,
                            modulus_int_preinverse<5> >,
```

```
                    modular_fixed<unsigned char, 2> > F2;
  ffe_2< F2, int, 3 > a;
  ffe_2< F2, int, 5 > b;
  return 0;
}
```

The variable `a` will represent an element of $\mathbb{F}_{2^3}$ while `b` will represent an element of $\mathbb{F}_{2^5}$. The `ffe_2` class is template with four arguments.

```
template< typename M, typename I, nat d,
          typename V= typename Ffe_2_variant(M,I,d) >
class ffe_2 {
  ...
};
```

The first argument `M` indicates the type chosen for the base field $\mathbb{F}_2$. Here it is the `modular` class with a fixed modulus. The second argument is the `C++` type that will be used to represents the elements of the finite field. The third argument indicates the degree of the extension over $\mathbb{F}_2$,

$$\mathtt{d} = [\mathtt{ffe\_2} < \mathtt{M}, \mathtt{I}, \mathtt{d} > : \mathbb{F}_2].$$

The fourth argument is the variant which has a default value. We list here all the possible variants for `ffe_2`.

`template< nat d > struct ffe_2_naive;`
: is the default variant. All the arithmetic is done with schoolbook algorithms. The argument `d` is the degree of the extension over $\mathbb{F}_2$. It must have the same value as the third argument of the `ffe_2` class.

`template< nat d > struct ffe_2_table;`
: is a variant where precomputations are done. The multiplication and inverse tables are computed. These two operations consist then of reading the wanted value in memory. The argument `d` is the degree of the extension over $\mathbb{F}_2$. It must have the same value as the third argument of the `ffe_2` class.

`template< nat d > struct ffe_2_mpfq;`
: is the variant where the arithmetic of the finite field is done by the `mpfq` library [GT06]. If you want to use this variant and if the `mpfq` library is not present on your system `Mathemagix` will automatically use the naive variant. The argument `d` is the degree of the extension over $\mathbb{F}_2$. It must have the same value as the third argument of the `ffe_2` class.

7.5 The quintix package

The quintix package contains many functions and interfaces that I test. They concern Galois rings, pseudo random generators, Reed-Solomon codes and linear algebra over Galois rings. In this section we describe only the implementation of the arithmetic of Galois rings and the implementation of the univariate polynomial root finding algorithms of [BLQ11].

7.5.1 Prime Galois rings

We begin with the construction of prime fields. Choose a prime integer and denote by p the corresponding variable in C++ of type I. We let r designate an integer of type nat. Prime Galois rings can be built, like prime finite fields, with the modular class. To take into account the p-adic structure of Galois rings a dedicated variant for modulus is proposed.

```
#include <numerix/modular.hpp>
#include <quintix/modulus_power.hpp>

int main(void) {
  I p;
  nat r;
  modulus< I, modulus_power< I, I, V > > mod= binpow (p, r);
  modular< modulus< I, modulus_power< I, I, V > > > a;
  set_modulus (a, mod);
  return 0;
}
```

Here we have

$$\text{“a} \in \mathbb{Z}/\text{p}^{\text{r}}\mathbb{Z}.\text{”}$$

The modulus_power structure is template and takes three arguments.

```
template< typename C, typename T, typename V= typename Modulus_variant
(C) > struct modulus_power :  V {};
```

The C argument indicates the integer type for the elements of the prime Galois ring, typically a type large enough to hold p^{r}. The T argument is the integer type of the prime p. The optional V argument is a valid variant for the modulus listed in Subsection 7.4.1.

7.5.2 Extensions of Galois rings

Galois rings behave almost like finite fields and their implementation is somehow similar to finite fields. The C++ class that represents a Galois ring extension is the gre class.

```
#include <basix/list.hpp>
#include <numerix/modular.hpp>
#include <quintix/gre_naive.hpp>
#include <quintix/modulus_power.hpp>
#include <quintix/residue_field.hpp>
#include <finitefieldz/ffe.hpp>

template< typename M, typename V= typename Gre_variant(M) >
class gre {
  ...
};
```

As before the `V` represents the variant for Galois rings which has a default value and thus need not be specified. The `M` argument is the base field. In fact the type `gre` represents an extension of the base field designated by the type `M`. For example, suppose that `M` is any finite field.

```
#include <quintix/gre.hpp>

int main(void) {
  gre< gre< gre< M > > > a, b;
  return 0;
}
```

The elements `a` and `b` lies in an extension of an extension of an extension of the finite field `M`. As often, we do not specify any variant and let `Mathemagix` decides what variant to use. As in the case of finite fields, the `set_defining_polynomial` can be used to specify the of the extension "`gre` $<$ `M`, `V` $>$/`M`". Suppose that `M` is $\mathbb{Z}/5^2\mathbb{Z}$, then one can make an extension of degree 3 with $X^3 + X + 1$.

```
#include <quintix/gre.hpp>
#include <quintix/polynomial.hpp>

int main(void) {
  polynomial< M > X= polynomial< M > (M (1), 1);
  polynomial< M > phi= (X * X * X) + X + 1;
  gre< M >::set_defining_polynomial (phi);
  return 0;
}
```

The constructed Galois ring is then

$$\text{“}\texttt{gre} < \texttt{M}, \texttt{V} > = \frac{\mathbb{Z}/5^2\mathbb{Z}[X]}{(\texttt{phi}(X))}\text{”}.$$

Global and local variants are proposed as for finite fields. Suppose that `M` represents $\mathbb{Z}/5^2\mathbb{Z}$.

```
#include <quintix/gre.hpp>
#include <quintix/polynomial.hpp>

int main(void) {
  polynomial< M > X= polynomial< M > (M (1), 1);
  polynomial< M > P= (X * X) + X + 1;
  polynomial< M > Q= (X * X * X) + X + 1;
  polynomial< M > R= binpow (X, 7) + X + 1;

  gre< M > a;
  gre< M, gre_naive > b;
  set_defining_polynomial (a, P);
  gre< M, gre_naive_local > c, d;
  set_defining_polynomial (c, Q);
  set_defining_polynomial (d, R);
  return 0;
}
```

The a and b variables have the same type gre< M, gre_naive > with the global, default variant. They live in the same extension of $\mathbb{Z}/5^2\mathbb{Z}$,

$$\text{“a, b} \in \frac{\mathbb{Z}/5^2\mathbb{Z}[X]}{(\mathtt{P}(X))}\text{”}.$$

The c and d variables have the same type with the local variant gre_naive_local. They do not, *a priori*, live in two different extensions

$$\text{“c} \in \frac{\mathbb{Z}/5^2\mathbb{Z}[X]}{(\mathtt{Q}(X))} \text{ and d} \in \frac{\mathbb{Z}/5^2\mathbb{Z}[X]}{(\mathtt{R}(X))}\text{”}.$$

It is recommended to use only the non-static versions of functions as they also work for variables of type gre< M, gre_naive >. The code produced following this guideline will then be compatible for local and global variant and it will be more generic. We list here some functions to manipulate Galois rings extensions.

`polynomial< M > get_defining_polynomial (gre< M, V >& x);`
returns the defining polynomial of the extension where x lives.

`void set_defining_polynomial (gre< M, V >& x, polynomial< M, W >& p);`
sets the polynomial for the extension where x lives. If x is of type gre< M, gre_naive > then all variables of type gre< M. gre_naive > will live in the same extension. If however x is of type gre< M, gre_naive_local > then only x will live in the extension defined by the polynomial p.

`nat get_extension_degree (gre< M, V >& x);`
returns the extension degree "[`gre< M, V >` : `M`]".

`void set_extension_degree (gre< M, V >& x, nat d);`
sets the degree of the extension where `x` lives to `d`.

`integer get_characteristic (gre< M, V >& x);`
returns the characteristic of the field where `x` lives.

`nat get_degree_over_prime_field (gre< M, V >& x);`
returns the degree of the extension where `x` lives over the prime field. If the prime field is $\mathbb{F}_p$ then it returns "[`gre< M, V >` : $\mathbb{F}_p$]" even if `M` $\neq \mathbb{F}_p$.

A Galois ring has a p-adic structure as it is a quotient ring of an unramified extension of $\mathbb{Z}_p$. Let $\mathrm{GR}(p^r, s)$ be a Galois ring and let

$$a = \sum_{i=0}^{r-1} a_i p^i.$$

We call *left shift* the map

$$\begin{array}{ccc} \mathrm{GR}(p^r,s) & \longrightarrow & \mathrm{GR}(p^r,s) \\ \sum_{i=0}^{r-1} a_i p^i & \longmapsto & \sum_{i=1}^{r-1} a_i p^{i-1}, \end{array}$$

and *right shift* the map

$$\begin{array}{ccc} \mathrm{GR}(p^r,s) & \longrightarrow & \mathrm{GR}(p^r,s) \\ a & \longmapsto & pa. \end{array}$$

The left and right shifts are implemented by the following functions:

`gre< M, V > lshiftz(const gre< M, V >& a, const nat& shift= 1);`
returns the left shift of `a`.

`gre< M, V > rshiftz(const gre< M, V >& a, const nat& shift= 1);`
returns the right shift of `a`.

It also possible to obtain the filtration of any element in Galois rings. Given any nonzero element a of a Galois ring $\mathrm{GR}(p^r, s)$, its filtration is defined to the valuation of any preimage of a in $\mathbb{Z}_{p^s}$. The filtration of $0 \in \mathrm{GR}(p^r, s)$ is set to $+\infty$.

`template<typename GR> nat get_valuation (const GR& a);`
returns the filtration of `a`.

Galois rings are local rings and computations need to be made in their residue field, see Chapters 1 to 4. A C++ helper structure is provided to obtain the residue field, as well as functions for computing residue images and preimages. Suppose that M is a Galois ring.

```
#include <quintix/gre.hpp>

int main(void) {
  typedef residue_field_helper< M >::F F;
  typedef gre< M > GR;
  typedef residue_field_helper< gre < M > > helper;
  typedef helper::F GF;
  GR a, b;
  GF abar, bbar;
  ...
  abar= helper::residue (a);
  b= helper::preimage (bbar);
  return 0;
}
```

This situation can be summed up with the following commutative diagram.

$$\begin{array}{ccccc} \texttt{GR} & \xrightarrow{\varphi} & \texttt{GF} & \longrightarrow & 0 \\ \uparrow & & \uparrow & & \\ \texttt{M} & \longrightarrow & \texttt{F} & \longrightarrow & 0 \\ \uparrow & & \uparrow & & \\ 0 & & 0 & & \end{array} \qquad \text{and } \texttt{abar} = \varphi(\texttt{a}), \texttt{bbar} = \varphi(\texttt{b}).$$

Let GR be a Galois ring and GF its residue field. The residue_field_helper contains helpful functions to compute residue field elements and their preimages. GR, the canonical surjection

Listing 7.5: quintix/include/quintix/residue_field.hpp

```
#include <quintix/modulus_power.hpp>
#include <numerix/modular_int.hpp>
#include <numerix/modular_integer.hpp>
#include <finitefieldz/ffe.hpp>
#include <quintix/modulus_power.hpp>
#include <algebramix/series.hpp>

template< typename FR >
struct residue_field_helper {
  typedef ... F;

  static inline
```

```
  F residue (const FR& a) {
    ...
  }

  static inline
  FR preimage (const F& a) {
    ...
  }
};
```

We have that

- "F = GF",
- residue : FR $\longrightarrow$ F is the canonical surjection and
- preimage : F $\longrightarrow$ FR computes preimages of elements of F.

7.5.3 Galois rings of characteristic 2^r

Prime Galois rings of characteristic 2^r can be efficiently implemented. The reduction modulo 2^r becomes a bitmask and a dedicated variant is provided.

Listing 7.6: quintix/include/quintix/gre_2.hpp

```
#include <numerix/modular.hpp>
#include <quintix/gre_naive.hpp>
#include <quintix/residue_field.hpp>
#include <finitefieldz/ffe.hpp>

template< nat d, typename I= nat >
class gre_2 {
  ...
};
```

The d template argument is the precision of the prime Galois ring gre_2< d, I > or gre_2 $<$ d, I $>$ $= \mathbb{Z}/2^d\mathbb{Z}$. The I argument indicates the integer type for to represent the elements of the Galois ring. Typically when d < 32 one can take I = int and I = integer otherwise.

Of course extensions of gre_2< d, I > can be made in the same way as in Subsection 7.5.2.

```
#include <quintix/gre_2.hpp>
#include <algebramix/polynomial.hpp>

int main(void) {
  typedef gre_2< 10, int > GR_2_10;
```

```
  polynomial< GR_2_10 > X=
        polynomial< GR_2_10 > (GR_2_10 (1), 1);
  typedef gre< GR_2_10 > GR;
  GR a;
  set_defining_polynomial (a, (X * X) + X + 1);
  return 0;
}
```

An extension of degree 2 is build with a prime Galois ring `gre_2< 10, int >` and we have

$$\text{“a} \in \frac{\texttt{gre\_2} < 10, \texttt{int} > [X]}{(X^2 + X + 1)}\text{”}.$$

7.5.4 Implementation of univariate root finding over Galois rings

The algorithms given in Chapter 2 have been implemented within the `quintix` package. The algorithms are valid over Galois rings. They have been implemented for Galois rings in the `quintix/include/quintix/solver_gre.hpp` file and for truncated power series in the `quintix/include/quintix/solver_dim1.hpp` file. The interface is very simple. The function returning the wanted roots is a static member of a structure which is template, allowing the user to choose between the naive algorithm (Algorithm 9) and the semifast algorithm (Algorithm 13).

```
#include <quintix/gre.hpp>
#include <quintix/solver_gre.hpp>
#include <quintix/solver_ffe.hpp>

int main(void) {
  polynomial< M > f;
  ...
  typedef solver_ffe< polynomial_berlekamp_naive > solver1;
  typedef solver_ffe< roots_tiny_field >           solver2;
  mmout << gre_roots< solver1 >::roots (f) << "\n";
  mmout << gre_roots< solver2 >::roots (f) << "\n";
  return 0;
}
```

The root finding algorithm over Galois rings needs a root finding algorithm over their residue fields. The `solver_ffe` structure indicates which root finding algorithm is to be used. The static `roots` function of the `gre_roots` structure is called and returns a vector of pairs. Each pair designates a class of roots for the polynomial `f` (Theorem 93 of Chapter 2).

```
template< typename Impl > struct solver_ffe;
```

contains the root finding algorithm for finite fields. The `Impl` argument indicates which algorithm to use. An exhaustive search can be performed when the finite field has a "small" cardinality with `roots_tiny_field`. The Berlekamp root finding algorithm can be used with `polynomial_berlekamp_naive`

`template< typename Impl, typename V= gre_roots_naive > struct gre_roots;`
contains the function that implements the root finding algorithm over Galois rings. The `Impl` argument indicates which root finding algorithm to use over the residue field. The optional `V` argument indicates the algorithm variant. The naive algorithm (Algorithm 9) can be used with `gre_roots_naive` while the semifast algorithm (Algorithm 13) is used with `gre_roots_hensel`. By default the naive variant is used.

`static vector< pair< GR, nat > > roots (const polynomial< GR, PV >& f);`
is the function inside the `gre_roots` structure that returns the roots of `f` $\in$ `GR`$[X]$. Each pair designates a class of roots for the polynomial `f` (Theorem 93 of Chapter 2).

Unlike the default naive variant, the semifast variant `gre_roots_hensel` needs a root finding function over the residue field that also returns the multiplicities of the roots. Therefore a dedicated structure must be used in this case.

```
#include <quintix/gre.hpp>
#include <quintix/solver_gre.hpp>
#include <quintix/ffe_roots_tiny_field.hpp>
#include <finitefieldz/berlekamp.hpp>
#include <quintix/solver_ffe.hpp>

int main(void) {
  polynomial< M > f;
  ...
  typedef solver_ffe_mult< polynomial_berlekamp_naive >
                                                solver1;
  typedef solver_ffe_mult< roots_tiny_field > solver2;
  mmout << gre_roots< solver1, gre_roots_hensel >::roots (f)
        << "\n";
  mmout << gre_roots< solver2, gre_roots_hensel >::roots (f)
        << "\n";
  return 0;
}
```

`template< typename Impl > struct solver_ffe_mult;`
contains the root finding algorithm for finite fields which also returns their multiplicities needed by the `gre_roots_hensel` variant. The `Impl` argument indicates which

algorithm to use. An exhaustive search can be performed when the finite field has a "small" cardinality with `roots_tiny_field`. The Berlekamp root finding algorithm can be used with `polynomial_berlekamp_naive`

The algorithms of Chapter 2 have also been implemented for truncated power series rings. The interface is exactly the same except for the name of structures and functions. They are prefixed by "`dim1`" instead of by "`gre`".

```
#include <quintix/gre.hpp>
#include <quintix/solver_gre.hpp>
#include <quintix/ffe_roots_tiny_field.hpp>
#include <finitefieldz/berlekamp.hpp>
#include <quintix/solver_ffe.hpp>

int main(void) {
  polynomial< M > f;
  ...

  typedef solver_ffe< polynomial_berlekamp_naive > solver1;
  typedef solver_ffe< roots_tiny_field >           solver2;

  mmout << dim1_roots< solver1 >::roots (f) << "\n";
  mmout << dim1_roots< solver2 >::roots (f) << "\n";

  typedef solver_ffe_mult< polynomial_berlekamp_naive >
                                                  solver3;
  typedef solver_ffe_mult< roots_tiny_field > solver4;
  mmout << dim1_roots< solver3,
                       dim1_roots_hensel >::roots (f)
        << "\n";
  mmout << dim1_roots< solver4,
                       dim1_roots_hensel >::roots (f)
        << "\n";

  return 0;
}
```

`template< typename Impl, typename V= dim1_roots_naive > struct dim1_roots;`

contains the function that implements the root finding algorithm over truncated power series rings. The `Impl` argument indicates which root finding algorithm to use over the residue field. The optional `V` argument indicates the algorithm variant. The naive algorithm (Algorithm 9) can be used with `dim1_roots_naive` while the

semifast algorithm (Algorithm 13) is used with `dim1_roots_hensel`. By default the naive variant is used.

```
static vector< pair< R, nat > > roots (const polynomial< R, PV >& f);
```

is the function inside the `dim1_roots` structure that returns the roots of `f` $\in$ `R`$[X]$. Each pair designates a class of roots for the polynomial `f` (Theorem 93 of Chapter 2).

The semifast variant needs a Hensel lifting which I have implemented with the help of Jérémy Berthomieu in the `quintix/include/quintix/homogeneous_hensel.hpp` file. We recall the definitions of Section 2.2.

Let R be any $\mathbb{N}$-graded ring. Recall that $K := \operatorname{Quot} R$ represents the *total field of fractions* of R. Since R is supposed to be complete, so is K, and we still write ν for the extension of the valuation from R to K. Any element a of K can be uniquely written as the sum $\sum_{i \geqslant \nu(a)} [a]_i$, where $[a]_i$ is 0 or has valuation i and is the quotient of two homogeneous elements in R. For any $i \in \mathbb{Z}$, we write K_i for the set of the elements $a \in K$ such that either a is 0 or a has a single component of valuation i, which means that $a = [a]_i$. The subset of the elements of K of valuation at least i is written O_i.

For any polynomial $F(x) = \sum_{l=0}^{d} F_l x^l \in K[x]$ of degree d, and any $w \in \mathbb{Z}$, we write $[F]_{i,w}$ for the polynomial

$$[F]_{i,w} := \sum_{l=0}^{d} [F_l]_{i-wl} x^l,$$

and call it the *w-homogeneous component of w-valuation i* of F. In addition, the expression $[F]_{j \ldots j+k,w}$ is used to represent the sum $\sum_{l=0}^{k-1} [F]_{j+l,w}$. Remark that if $a \in K$ has valuation at least w then $[F]_{i,w}(a)$ has valuation at least i. Finally the quantity $\nu_w(F)$, called the *w-valuation* of F, stands for the first index $i \in \mathbb{Z}$ such that $[F]_{i,w}$ is nonzero, with the convention that $\nu_w(0) := +\infty$.

The `hensel` function is a static member of the `homogeneous_hensel_helper` structure. It can be accessed directly with the `homogeneous_hensel` function.

```
template< typename C, typename V > vector< polynomial< C, V > >
homogeneous_hensel (const polynomial< C, V >& f, const vector<
polynomial< C, V > >& H, nat w, nat n);
```

returns the lifting of the `w`-homogeneous polynomials up to precision `n` of the factors contained in `H` such that $[\mathtt{f}]_{j,w} = [\mathtt{H}_1 \times \mathtt{H}_2 \times \cdots \times \mathtt{H}_s]_{j,w}$ where j is the sum of all the w-valuation of the polynomials $\mathtt{H}_i$ of `H`. It is an implementation of Algorithm 12.

Chapter 8

The decoding Library for List Decoding

The first section of this chapter constitutes an accepted extended abstract at ISSAC (International Symposium on Symbolic and Algebraic Computation) 2012. The other sections gives more details about the library.

8.1 Overview of `decoding`

8.1.1 Introduction and motivation

Reed-Solomon (RS) codes form an important and well-studied family of codes. They were first proposed in 1960 by Reed and Solomon in their original paper [RS60]. They are widely used in practice [WB99]. RS codes can be efficiently unique decoded [Gao02] and [Jus76]. Sudan's 1997 breakthrough on list decoding of RS codes [Sud97b], further improved by Guruswami and Sudan in [GS98], showed that RS codes are list decodable up to the Johnson bound in polynomial time. DECODING is a C library whose main goal is to implement as efficiently as possible the Guruswami-Sudan algorithm. It is written in C89 and is stand-alone.

The Guruswami-Sudan algorithm

The DECODING library is devoted to algorithms concerning the Guruswami-Sudan list decoding scheme and does not limit itself to finite fields as it is often the case in coding theory. Let us fix a finite ring with identity A not necessarily commutative. The Guruswami-Sudan algorithm has two main steps. The first one (interpolation step) consists in finding a "curve" of equation $Q(X, Y) = 0$ in A^2 which passes through given points with certain multiplicities. The second step (root-finding) finds the roots of $Q(X, Y)$ seen in $(A[X])[Y]$.

The interpolation step dominates the cost of the whole Guruswami-Sudan algorithm. Many methods have been proposed but without any available implementations or com-

parisons to other ones. A few timings of the Guruswami-Sudan algorithm can be found, but again without the corresponding implementation.

8.1.2 The implementation

To the knowledge of the author no implementation of the Guruswami-Sudan algorithm has been proposed. The only available implementation is constituted by a set of C++ functions, not directly accessible, inside PERCY++ [Gol07b] whose purpose is not error correction and which does not use fast algorithms for dense bivariate polynomials.

The algorithms provided by decoding

The implemented algorithm for interpolation is a variant of the Koetter algorithm [McE03] in the `include/decoding/algos/koetter.c` file. It uses polynomial arithmetic with fast bivariate shifting (computation of $Q(X + x_0, Y + y_0)$ where $(x_0, y_0) \in A^2$) in `include/decoding/algos/dbpol_shift_fast.c` and fast univariate shifting (computation of $f(X + x_0)$ where $f \in A[X]$ and $x_0 \in A$) in `include/decoding/algos/upol_shift_fast.c`. Specific variants of these algorithms for commutative rings of characteristic 2 are also present in the same files.

The second step (root-finding) implemented in the library is a variant of the Roth and Ruckenstein algorithm [RR98] and the naive algorithm of [BLQ11]. It is in `include/decoding/algos/dbpol_Xroots.c`.

The design of decoding

The DECODING library is designed to be easy to use in a C or C++ program. One of its particularities is to use the C preprocessor to generate algorithms for a ring (generally a finite field) which must be provided by the end-user. Therefore efficient libraries like MPFQ [GT06] can be used by the end-user. For the sake of completeness, some finite fields are provided by default.

Error correcting codes are often regarded over finite fields, in particular $\mathbb{F}_2$, together with the classical Hamming distance. But other distances, like the Lee distance, are better suited for some applications. Usually the Lee distance is needed for codes over Galois rings. Error correcting codes over the ring of matrices over a finite field or a finite commutative ring are also considered for example in [OSB12]. Therefore DECODING proposes generic algorithms whenever possible. This flexibility is needed when studying codes over Galois rings for example where the end-user needs to manipulate codes over a Galois ring and its residue field at the same time.

Although DECODING proposes certain fast bivariate polynomial algorithms, it is not its goal to propose fast algorithms for univariate and bivariate polynomial multiplication. In fact, DECODING is designed to be used in conjunction with other efficient libraries like GMP [Gra91], NTL [Sho90] or FLINT [Har10]. For the sake of completeness DECODING provides these algorithms in their "schoolbook" form but it is recommended, for efficiency, to use external libraries.

A very simple mechanism using the C preprocessor allows one to override the default generic algorithms proposed by DECODING. For example see at the `include/decoding/rings/GF5.c` file which implements the finite field $\mathbb{F}_5$. It shows how to replace the univariate polynomial root finding over $\mathbb{F}_5$. All C macros that control this mechanism are in the `include/decoding/ring_reset.h` file.

8.1.3 Presentation

The DECODING library is the first library which proposes a flexible and efficient way to implement algorithms related to the Guruswami-Sudan decoding scheme. It can be used with efficient external libraries to obtain more efficient implementations of Guruswami-Sudan related algorithms.

I will first present quickly the history of the Guruswami-Sudan algorithm and show that it needs dense bivariate polynomials only available, not necessarily directly, in computer algebra systems such as MAGMA [BCP97] or MATHEMAGIX [H+02]. As error correcting codes are often used over binary fields, dedicated fast algorithms must be used. The bivariate polynomials appearing in the list decoding algorithms can have large degrees even when RS codes with small parameters are considered. Hence fast algorithms are needed.

I will then present the flexibility of DECODING, needed to obtain efficient algorithms over several finite rings and fields.

- It is easy to replace a key algorithm, such as univariate polynomial multiplication or univariate polynomial root-finding, by a very efficient one provided by an external library such as FLINT or NTL for example.

- It is easy to choose a finite ring or a finite field, or even to use different rings at the same time in order to implement algorithms related to RS codes over Galois rings. The DECODING library is not restricted to mathematical object whose binary representation holds in a single machine word. It requires no supplementary efforts to use, for example, multiple precision integers from GMP or large binary fields from MPFQ.

Finally, I will present the provided algorithms concerning dense bivariate polynomials and their applications to list decoding.

8.2 More details on `decoding`

8.2.1 The directory tree of `decoding`

The `decoding` library follows the general principal of `GMP` [Gra91]. There are two main sets of functions.

- The functions in the `include/decoding/lowlevels` directory are low levels functions and do not provide a coherent calling interface. Their names are prefixed

by `ll`. They aim at implementing algorithms as efficiently as possible. They are the equivalent of the `mpn` functions from `GMP`. Each file contains a coherent group of functions. For example, the `include/decoding/lowlevels/vec.c` contains all the vectors manipulation functions. When using the `ll` functions, memory has to be properly managed by hand. This can be tricky as no verification is performed.

- The functions in the `include/decoding/sugar` directory form a coherent calling interface and are to be called for a normal use of the library. On the contrary to the `ll` functions they perform verifications on their input and take care of all the memory needed. They are the equivalent of the `mpz` functions from `GMP`. Each file contains a coherent group of functions. For example, the `include/decoding/sugar/vec.c` contains all the vectors manipulation functions which will manage the memory and call the corresponding `ll` functions of `include/decoding/lowlevels/vec.c`.

The `include/decoding/rings` directory contains the implementation of the rings (finite rings, finite fields) provided by `decoding`. Each file corresponds to a single ring or a family of rings. For example `include/decoding/rings/gfp_word.c` implements all the finite fields $\mathbb{F}_p$ where p holds within a machine word. The `tests`, `benchs` and `samples` directories contains respectively test files, bench files and example files for the library. The `doc` directory contains the documentation in two different formats.

- The `doc/man3` directory contains UNIX manpages.
- The `doc/html` directory contains HTML pages.

The documentation is generated from files in the `doc` directory and from the comments contained in the `.c` files. Finally the `utils` directory contains some utilities for building the library.

The library depends on two external libraries.

- The `GMP` library is used for the arithmetic of univariate and bivariate polynomials over the rings $\mathbb{Z}/n\mathbb{Z}$.
- The `mpfq` library is used for the arithmetic of the finite fields $\mathbb{F}_{2^n}$, $2 \leq n \leq 255$.

You need to install or build `GMP` and `mpfq` in order to build `decoding`. To build the library it is sufficient to type `make`. The `Makefile` will detect the best options to pass to the compiler and then the `libdecoding.a` will be built.

8.2.2 The internals of the library

The `libdecoding.a` contains in fact a few functions about the computation of the parameters of the Guruswami-Sudan algorithm, they do not require optimizations. All the other `C` functions are not compiled, they must be included (`#include`) in the program you write. This has an advantage, the compiler sees the code as a whole and is therefore able to perform some optimizations such as inlining small functions. We present an

example of this mechanism. This is a *simplified* example which does *not* correspond to the actual code of the library.

Listing 8.1: `decoding.h`

```
/* macro concatenation */
#define __C2(a,b) a##b
#define __C(a,b)  __C2(a,b)

/* Addition functions */
#define R_add     __C(R,_add)
#define R_vec_add __C(R,_vec_add)
```

Listing 8.2: `gfp.c`

```
#define R RING_NAME
typedef R int[1];

void R_add(R c,R a,R b) {
  c[0] = a[0] + b[0] % p;
}
```

Listing 8.3: `gf2n.c`

```
#define R RING_NAME
typedef R int[1];

void R_add(R c,R a,R b) {
  c[0] = a[0] ^ b[0];
}
```

Listing 8.4: `vec.c`

```
void R_vec_add(R *r,R *a,R *b,int n) {
  int i;
  for( i = 0 ; i < n ; i++ ) {
    R_add(r[i],a[i],b[i]);
  }
}

#undef RING_NAME
#undef R
```

The `decoding.h`, `gfp.c`, `gf2n.c` and `vec.c` files would be provided by the library. The following listing corresponds to a file written by a user of `decoding`.

Listing 8.5: program.c

```
#include <decoding.h>

#define RING_NAME gfp
#include <gfp.c>
#include <vec.c>

#define RING_NAME gf2n
#include "gf2n.c"
#include "vec.c"

int main(void) {
  gfp a[5],b[5],c[5];
  gf2n u[5],v[6],w[5];

  gfp_vec_add(a,b,c);
  gf2n_vec_add(u,v,w);

  return 0;
}
```

To better understand what happens when the compiler processes program.c, it is interesting to see the output of the C preprocessor.

Listing 8.6: Output of "cpp program.c"

```
typedef gfp int[1];

void gfp_add(gfp c,gfp a,gfp b) {
  c[0] = (a[0] + b[0]) % p;
}

void gfp_vec_add(gfp *r,gfp *a,gfp *b,int n) {
  int i;
  for( i = 0 ; i < n ; i++ ) {
    gfp_add(r[i],a[i],b[i]);
  }
}

typedef gf2n int[1];

void gf2n_add(gf2n c,gf2n a,gf2n b) {
  c[0] = a[0] ^ b[0];
}
```

```
void gf2n_vec_add(gf2n *r,gf2n *a,gf2n *b,int n) {
  int i;
  for( i = 0 ; i < n ; i++ ) {
    gf2n_add(r[i],a[i],b[i]);
  }
}

int main(void) {
  gfp a[5],b[5],c[5];
  gf2n u[5],v[6],w[5];

  gfp_vec_add(a,b,c);
  gf2n_vec_add(u,v,w);

  return 0;
}
```

The `R_add` function is provided by both `gfp.c` and `gf2n.c`. This function implements the addition of two elements from the ring `R`. In `gfp.c` it corresponds to the usual addition modulo `p` while in `gf2n.c` it corresponds to the addition in $\mathbb{F}_{2^n}$, which can be implemented with a XOR. Within the `vec.c` file, we do not know which ring we manipulate so we use the `R_add` function to add two elements from `R` and then obtain a generic component-wise addition for vectors. This mechanism allows one to write algorithms with a limited genericity to the choice of the underlying ring and the user can benefit from a unified interface.

When the user wants to use a ring, he must provide a name for the ring using the `RING_NAME` macro. This name will also be a genuine `C` type representing an element of the ring.

```
#define RING_NAME gfp
#include <gfp.c>
```

As the `R_add` token is defined as a macro in `decoding.h` it will be replaced by `RING_NAME_add` and, as `RING_NAME` has the value `gfp`, will be further replaced by `gfp_add`. The same holds for `R_vec_add`. Therefore `gfp_add`, `gfp_vec_add` and `gf2n_add`, `gf2n_vec_add` will be generated by the preprocessor and the user will be able to use both $\mathbb{F}_p$ and $\mathbb{F}_{2^n}$ at the same time.

The macros

```
#define R_add      __C(R,_add)
#define R_vec_add  __C(R,_vec_add)
```

are declared in a separate file from the actual implementation of the additions. Therefore the code contained in all the `.c` files contained in the `include/decoding/lowlevels` and `include/decoding/sugar` directories is valid `C` code without macros and can be

used by itself. A direct consequence is that error and warning messages printed by the compiler are quite simple to decipher.

8.2.3 Customization of the library

It is possible and very simple to customize the library. Suppose that you want to unroll the loops for vector addition for efficiency. You can achieve this by giving to the `MY_VEC_ADD` macro the name of the function you provide to replace vector addition.

Listing 8.7: Modified `program.c`

```
#include <decoding.h>

#define RING_NAME gfp
#include <gfp.c>

#define MY_VEC_ADD my_vec_add
void my_vec_add(R *r,R *a,R *b,int n) {
  int i,m;
  m = n % 4;
  for( i = 0 ; i < m ; i++ ) {
    R_add(r[i],a[i],b[i]);
  }
  for( ; i < n ; i += 4 ) {
    R_add(r[i    ],a[i    ],b[i    ]);
    R_add(r[i + 1],a[i + 1],b[i + 1]);
    R_add(r[i + 2],a[i + 2],b[i + 2]);
    R_add(r[i + 3],a[i + 3],b[i + 3]);
  }
}

#include <vec.c>

#define RING_NAME gf2n
#include "gf2n.c"
#include "vec.c"

int main(void) {
  gfp a[5],b[5],c[5];
  gf2n u[5],v[6],w[5];

  gfp_vec_add(a,b,c);
  gf2n_vec_add(u,v,w);

  return 0;
```

```
}
```

In order for this to work the library contains a modified version of `vec.c`.

Listing 8.8: Modified `vec.c`

```
void R_vec_add(R *r,R *a,R *b,int n) {
#ifdef MY_VEC_ADD
  MY_VEC_ADD(r,a,b,n);
#else
  int i;
  for( i = 0 ; i < n ; i++ ) {
    R_add(r[i],a[i],b[i]);
  }
#endif
}

#undef RING_NAME
#undef R
#ifdef MY_VEC_ADD
#undef MY_VEC_ADD
#endif
```

Many low level functions can be customized by the user including, among other, vector arithmetic, polynomial arithmetic and memory management functions. Notable examples are for finite fields of characteristic 2 whose implementations has to be completely different from prime fields in order to be efficient. By default the library provides *portable* and *as optimized as possible* (under the constraint of portability) `C` code. The customization macros allows a user to put his own (more efficient) code in a very simple way. All the currently implemented customization macros can be found in `include/decoding/ring_reset.h`. The example implementation of $\mathbb{F}_{17}$ in `include/decoding/rings/GF17.c` shows how to use customization macros.

8.2.4 Rings provided by default with the library

By default, `decoding` comes with several finite fields. Each ring is implemented in a separate file within the `include/decoding/rings` directory. To use a ring provided by the library, say `myring.c` you must do

```
#include <decoding/decoding.h>

#define RING_NAME rng
#include <decoding/rings/myring.c>
#include <decoding/algos.c>

int main(void) {
```

```
  ...

  return 0;
}
```

where `RING_NAME` is the name you want to give to the ring. It will also be a genuine `C` type representing an element of the ring. The include line

```
#include <decoding/algos.c>
```

imports all the code implementing vectors, polynomials, error correcting codes. The files `GF5.c` and `GF17.c` are examples of implementations of finite fields. They are not to be used if one wants optimized implementations. Here is a list of finite fields contained in `include/decoding/rings`.

Small prime fields

The file `gfp_word.c` implements all prime fields $\mathbb{F}_p$ whose characteristic holds within a machine word. Use `PRIME` to indicate the modulus of the finite field you want to construct.

```
#define PRIME 101
#include <decoding/rings/gfp_word.c>
#include <decoding/algos.c>
```

In the above example the name of the finite field will be `gf101`. You can of course use several prime fields in the same program.

```
#define PRIME 7
#include <decoding/rings/gfp_word.c>
#include <decoding/algos.c>

#define PRIME 11
#include <decoding/rings/gfp_word.c>
#include <decoding/algos.c>
```

It will then generate the code for the two finite fields $\mathbb{F}_7$ and $\mathbb{F}_{11}$. You will then have two types.

- `gf7` corresponding to $\mathbb{F}_7$.
- `gf11` corresponding to $\mathbb{F}_{11}$.

The prime number defining the field is accessible with the `R_characteristic` variable. For example we will have the following.

- `gf7_characteristic` $= 7$.

- gf11_characteristic = 11.

By default, the unsigned int C type is chosen to hold the elements of the field. You can specify a different genuine integer C type like unsigned short or unsigned long if you need with WORD_TYPE. You must choose an unsigned type.

```
#define WORD_TYPE unsigned long
#define PRIME 101
#include <decoding/rings/gfp_word.c>
#include <decoding/algos.c>
```

- void R_ring_init(void);
  Initializes the ring. A call to this function is mandatory before doing anything else.

- void R_ring_clear(void);
  Free the memory occupied by the ring. A call to this function is recommended when the ring is no more needed.

Dynamic small prime fields

The implementation in gfp_word.c is static. The characteristic is chosen and known at compile time. The user cannot change it at run time. The gfp_word_dynamic.c file implements exactly the same finite fields as gfp_word.c with the possibility of changing the characteristic at run time. It implements all prime fields $\mathbb{F}_p$ whose characteristic holds within a machine word.

```
#include <decoding/rings/gfp_word_dynamic.c>
#include <decoding/algos.c>
```

In the above example the name of the finite field will be gfp. You can of course use several prime fields in the same program as soon as they have different names.

```
#define RING_NAME fld1
#include <decoding/rings/gfp_word_dynamic.c>
#include <decoding/algos.c>

#define RING_NAME fld2
#include <decoding/rings/gfp_word_dynamic.c>
#include <decoding/algos.c>
```

The prime number defining the field is accessible with the R_characteristic variable. For example we will have access to the following variables:

- fld1_characteristic.

- fdl2_characteristic.

By default, the `unsigned int C` type is chosen to hold the elements of the field. You can specify a different genuine integer `C` type like `unsigned short` or `unsigned long` if you need with `WORD_TYPE`. You must choose an unsigned type.

```
#define WORD_TYPE unsigned long
#include <decoding/rings/gfp_word_dynamic.c>
#include <decoding/algos.c>
```

- `void R_ring_init(R_characteristic p);`
  Initialize the ring with prime p. A call to this function is mandatory before doing anything else.
- `void R_ring_clear(void);`
  Free the memory occupied by the ring. A call to this function is recommended when the ring is no more needed.

You can change the prime of the ring with successive calls to `R_ring_clear` and `R_ring_init`.

```
#include <decoding/decoding.h>
#define RING_NAME fld
#include <decoding/rings/gfp_word_dynamic.c>

...

int main(void) {
  fld_init_ring(101);
  ...
  fld_clear_ring();
  fld_init_ring(67);
  ...
  fld_clear_ring();
  return 0;
}
```

Finite fields of characteristic 2

The `mpfq_gf2n_wrapper.c` file implements all fields $\mathbb{F}_{2^n}$ with $2 \leq s \leq 255$. It is a wrapper to `mpfq` [GT06]. The `DEGREE` indicates which extension of $\mathbb{F}_2$ you want to use.

```
#define DEGREE 8
#include <decoding/rings/mpfq_gf2n_wrapper.c>
#include <decoding/algos.c>
```

In the above example the name of the finite field will be `gf2_8`. You can of course use several `mpfq` fields in the same program.

```
#define DEGREE 8
#include <decoding/rings/mpfq_gf2n_wrapper.c>
#include <decoding/algos.c>

#define DEGREE 64
#include <decoding/rings/mpfq_gf2n_wrapper.c>
#include <decoding/algos.c>
```

It will then generate the code for the two finite fields $\mathbb{F}_{2^8}$ and $\mathbb{F}_{2^{64}}$. You will then have two types.

- `gf2_8` corresponding to $\mathbb{F}_{2^8}$.
- `gf2_64` corresponding to $\mathbb{F}_{2^{64}}$.

The degree of the extension is accessible with the `R_ext_degree` variable. For example we will have the following.

- `gf2_8_ext_degree` $= 8$.
- `gf2_64_ext_degree` $= 64$.

8.2.5 Implemented algorithms

Algorithms in low levels functions

The Kötter algorithm [Köt96] has been implemented with polynomial shiftings computations instead of binomials computations in the `include/decoding/lowlevels/koetter*.c` files. It makes the Kötter algorithm faster. The polynomial shifting algorithm implemented in the `include/decoding/lowlevels/upol_shift*.c` and `include/decoding/lowlevels/dbpol_shift*.c` files follows the divide-and-conquer paradigm and can be found in [BLQ11, Lemma 14, page 12]. The latter algorithm needs quasi-linear univariate polynomial multiplication in order to be quasi-linear. The Kronecker substitution [Kro82] has been implemented for fields of characteristic $\neq 2$ in `include/decoding/lowlevels/upol_kronecker_word.c` and `include/decoding/generic/kronecker.c`.

The binomials computations are done with `GMP` in characteristic $\neq 2$ and a special variant has been written for characteristic 2. It can be found in `include/decoding/lowlevels/binomial.c`, `include/decoding/lowlevels/*_discrepancy.c`.

The `decoding` API

We list here a few of the "sugar" functions provided by `decoding`. They are the equivalent of `GMP` `mpz` functions, form a coherent calling interface and should be called by the

user of the library. Their implementation can be found in the `include/decoding/sugar` directory. The function names are prefixed by the name of the chosen ring.

- `void vec_init(vec r);`
  Initialize the vector `r`. This function must be called before anything else.

- `void vec_inits(vec r,...);`
  Initialize a NULL-terminated list of vectors. This function must be called before anything else.

- `void vec_clear(vec r);`
  Free the memory occupied by the vector `r`. You must not use `r` after a call to this function. If you want to use `r` you must make a call to `vec_init`.

- `void vec_clears(vec r,...);`
  Free a NULL-terminated list of vectors. You must not use the vectors after a call to this function. If you want to use some or all of the vectors you must make calls to `vec_init` or `vec_inits`.

- `void vec_zero(vec r);`
  Set all the components of `r` to zero.

- `void vec_append(vec r,R a);`
  Append `a` to `r`.

- `void vec_random(vec r);`
  Set all the components of `r` to random elements of the ring.

- `void vec_random_hamming_weight(vec r,int nr,int w);`
  Set `r` to be a random vector of length `nr` and Hamming weight `w`.

- `void vec_add(vec r,vec a,vec b);`
  Set `r` to `a + b`. If `a` and `b` do not have the same length the shortest vector is padded with zeros.

- `void vec_sub(vec r,vec a,vec b);`
  Set `r` to `a - b`. If `a` and `b` do not have the same length the shortest vector is padded with zeros.

- `void vec_mul_by_cte(vec r,vec a,R b);`
  Set `r` to `a * b`.

- `void vec_set_coeff(vec r,R a,int i);`
  Set the `i`-th coefficient of `r` to `a`.

- `void vec_get_coeff(R r,vec a,int i);`
  Set `r` to the `i`-th coefficient of `a`.

- `int vec_get_len(vec a);`
  Return the length of `a`.

- `int vec_hamming_weight(vec a);`
  Return the Hamming weight of `a`.

- `int vec_hamming_distance(vec a,vec b);`
  Return the Hamming distance between `a` and `b`. If `a` and `b` do not have the the same size then `-1` is returned.

- `int vec_copy(vec r,vec a);`
  Make a deep copy of `a` into `r`.

- `int vec_cmp(vec a,vec b);`
  Return `0` if and only if `a` equals `b`. If `a` and `b` do not have the same length they are not equal.

- `void vec_ptr_clear(vec *a,int n);`
  Clear the array `a` of vectors of size `n`.

- `void vec_print(FILE *out,vec_ptr a);`
  Print `a` into `out`.

- `void rs_code_init(rs_code rs,int n,int k);`
  Initialize `rs` of length `n` and dimension `k`. This function must be called before doing anything else with `rs`. The support of `rs` will be a zero vector.

- `void rs_code_init_with_support(rs_code rs,int n,int k);`
  Initialize `rs` of length `n` and dimension `k`. This function must be called before doing anything else with `rs`. `Decoding` will attempt to create the support of `rs`: it will first set the first coordinate to zero, then it will use `R_next` to set the other coordinates. Make sure that the ring you are using has enough elements. For example, if the ring is a prime field then the support will be set to `[0,1,2,3,...,n]`.

- `void rs_code_clear(rs_code rs);`
  Free the memory occupied by `rs`.

- `void rs_code_random(vec r,rs_code rs);`
  Set `r` to be a random codeword of `rs`.

- `int rs_code_sudan_koetter(vec **r,rs_code rs,vec y,int tau);`
  Given a Reed-Solomon code `rs` and a received word `y` with at most `tau` errors, return the number of codewords within distance `tau` of `y` and set `r` to an array containing the codewords. The Sudan algorithm is used, the interpolation step is done with the Kötter algorithm.

- `int rs_code_guruswami_sudan_koetter(vec **r,rs_code rs,vec y,int tau);`
  Given a Reed-Solomon code `rs` and a received word `y` with at most `tau` errors, return the number of codewords within distance `tau` of `y` and set `r` to an array containing the codewords. The Guruswami-Sudan algorithm is used, the interpolation step is done with the Kötter algorithm.

8.2.6 Timings

We present some timings of the Guruswami-Sudan algorithm done by `decoding` and compare them to a similar implementation from the `Percy++` library [Gol07b] although the main goal of `Percy++` is not to provide decoding algorithm but rather to provide a framework for Private Information Retrieval (PIR) [CGKS95, Gol07a]. The `C` and `C++` code for the timings of both `decoding` and `Percy++` can be found in `include/decoding/benchs`.

The timings were done on an Intel(R) Core(TM)2 CPU 6400 @ 2.13GHz, with 4Go of RAM, `GMP` 5.0.5, `mpfq` 1.0-rc3, `Percy++` 0.3 and `NTL` 5.5.2, compiled with gcc (GCC) 4.4.6 20120305.

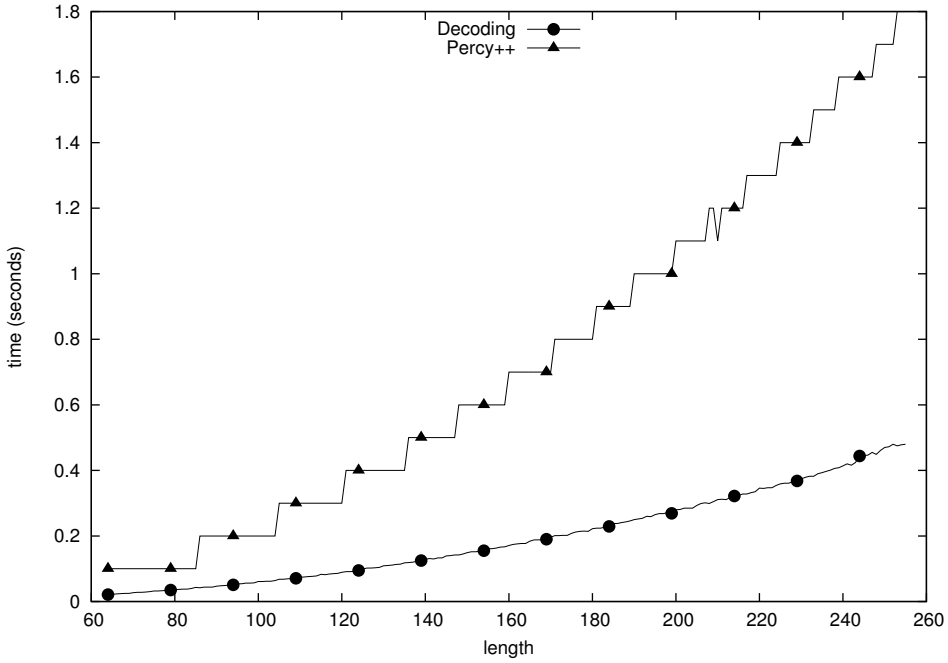

Figure 8.1: Timings over $\mathbb{F}_{257}$

The implementation of the Guruswami-Sudan algorithm in `Percy++` provides several interpolation algorithms. We have compared the `Percy++` Kötter implementation to our implementation which can be found in `include/decoding/lowlevels/koetter.c`. For both figure 8.1 and 8.2 the Guruswami-Sudan list decoding algorithm is applied for

Reed-Solomon codes with parameters $[n, \lfloor n/5 \rfloor]_{\mathbb{F}_{257}}$, to words having a number of errors corresponding to the multiplicity 2.

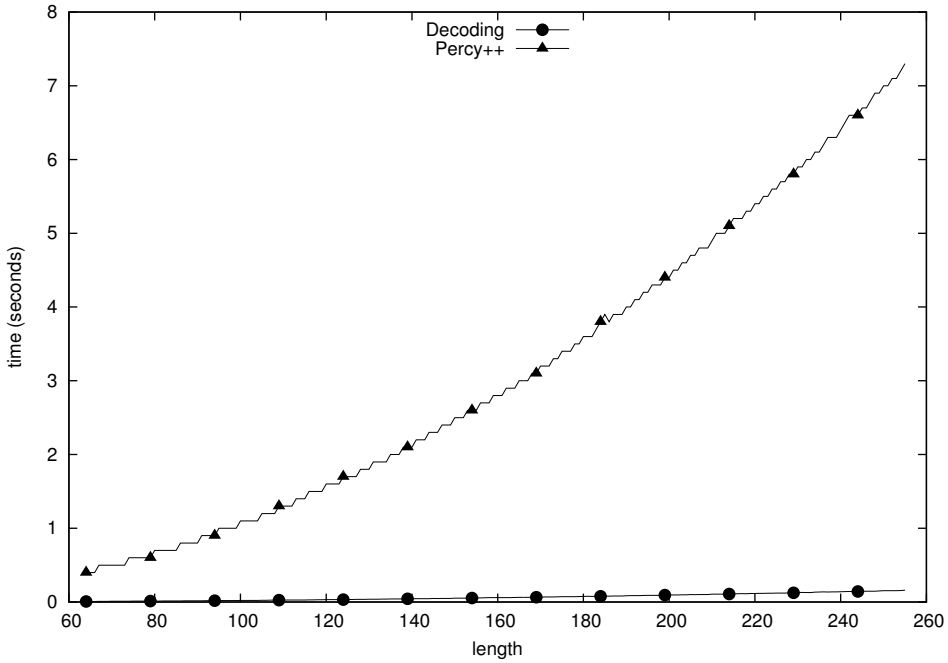

Figure 8.2: Timings over $\mathbb{F}_{2^8}$

Figure 8.1 shows the timings for Reed-Solomon codes over $\mathbb{F}_{257}$ where `decoding` is faster than `Percy++` by a factor of 4. Figure 8.2 shows the timings for Reed-Solomon codes over $\mathbb{F}_{2^8}$ where `decoding` is faster than `Percy++` by a factor of 45.

Bibliography

[ABC10] D. Augot, M. Barbier, and A. Couvreur. List-decoding of binary Goppa codes up to the binary Johnson bound. Research Report RR–7490, INRIA, 2010.

[ABC11] D. Augot, M. Barbier, and A. Couvreur. List-decoding of binary Goppa codes up to the binary Johnson bound. In *Information Theory Workshop (ITW), 2011 IEEE*, pages 229–233, oct 2011.

[AdT05] M. A. Armand and O. de Taisne. Multistage list decoding of generalized Reed-Solomon codes over Galois rings. *Communications Letters, IEEE*, 9(7):625–627, jul 2005.

[Ajt98] M. Ajtai. The shortest vector problem in l_2 is NP-hard for randomized reductions (extended abstract). In *Proceedings of the thirtieth annual ACM symposium on Theory of computing*, STOC'98, pages 10–19, New York, NY, USA, 1998. ACM.

[Ale05] M. Alekhnovich. Linear Diophantine equations over polynomials and soft decoding of Reed-Solomon codes. *IEEE Trans. Inform. Theory*, 51(7):2257–2265, 2005.

[ALRS92] S. Ar, R.J. Lipton, R. Rubinfeld, and M. Sudan. Reconstructing algebraic functions from mixed data. In *Foundations of Computer Science, 1992. Proceedings, 33rd Annual Symposium on*, pages 503–512, oct 1992.

[ALRS98] S. Ar, R.J. Lipton, R. Rubinfeld, and M. Sudan. Reconstructing Algebraic Functions from Mixed Data. *SIAM Journal of Computing*, 28(2):487–510, 1998.

[AM94] M.F. Atiyah and I.G. MacDonald. *Introduction to commutative algebra.* Addison-Wesley series in mathematics. Westview Press, 1994.

[Arm02] M. A. Armand. Efficient decoding of Reed-Solomon codes over $\mathbb{Z}_q$ based on remainder polynomials. *WSEAS Transactions on Communications*, 1(1):116–121, 2002.

[Arm04] M. A. Armand. Improved list decoding of generalized Reed-Solomon and alternant codes over rings. In *IEEE International Symposium on Information Theory 2004 (ISIT 2004)*, page 384, 2004.

[Arm05a] M. A. Armand. Improved list decoding of generalized Reed-Solomon and alternant codes over Galois rings. *IEEE Trans. Inform. Theory*, 51(2):728–733, feb 2005.

[Arm05b] M. A. Armand. List decoding of generalized Reed-Solomon codes over commutative rings. *IEEE Trans. Inform. Theory*, 51(1):411–419, 2005.

[Arm05c] M. A. Armand. Solving the Welch-Berlekamp key equation over a Galois ring. In *WSEAS Transactions on Mathematics*, volume 4, pages 319–326, 2005.

[Arm10] M. A. Armand. A Note on Interleaved Reed-Solomon Codes Over Galois Rings. *IEEE Trans. Inform. Theory*, 56(4):1574–1581, april 2010.

[Aya10] A. Ayad. A lecture on the complexity of factoring polynomials over global fields. *International Mathematical Forum*, 5(10):477–486, 2010.

[AZ08] D. Augot and A. Zeh. On the Roth and Ruckenstein Equations for the Guruswami-Sudan Algorithm. In *IEEE International Symposium on Information Theory - ISIT 2008*, pages 2620–2624, Toronto, Canada, July 2008. IEEE.

[Bar06] K. Bartley. *Decoding algorithms for algebraic geometric codes over rings.* PhD thesis, University of Nebraska at Lincoln, Lincoln, NB, USA, 2006.

[BCGO09] T. Berger, P.-L. Cayrel, P. Gaborit, and A. Otmani. Reducing Key Length of the McEliece Cryptosystem. In *Proceedings of the 2nd International Conference on Cryptology in Africa: Progress in Cryptology*, AFRICACRYPT '09, pages 77–97, Berlin, Heidelberg, 2009. Springer-Verlag.

[BCP97] W. Bosma, J. Cannon, and C. Playoust. The Magma algebra system. I. The user language. *J. Symbolic Comput.*, 24(3-4):235–265, 1997. Computational algebra and number theory (London, 1993).

[BCQ12] M. Barbier, C. Chabot, and G. Quintin. On quasi-cyclic codes as a generalization of cyclic codes. *Finite Fields and Their Applications*, 18(5):904–919, 2012.

[Ber68] E.R. Berlekamp. *Algebraic Coding Theory.* McGraw Hill, New York, 1968.

[Ber84] E. R. Berlekamp. *Algebraic coding theory.* M-6. Aegean Park Press, 1984.

[BF01] E. Byrne and P. Fitzpatrick. Gröbner Bases over Galois Rings with an Application to Decoding Alternant Codes. *J. Symbolic Comput.*, 31(5):565–584, 2001.

[BF02] E. Byrne and P. Fitzpatrick. Hamming metric decoding of alternant codes over Galois rings. *IEEE Trans. Inform. Theory*, 48(3):683–694, mar 2002.

[BF12] J.-F. Biasse and C. Fieker. A polynomial time algorithm for computing the HNF of a module over the integers of a number field, 2012. `http://www.lix.polytechnique.fr/~biasse/papers/HNF_pol.pdf`.

[BGTZ08] R. Brent, P. Gaudry, E. Thomé, and P. Zimmermann. Faster multiplication in GF(2)[x]. In *Proceedings of the 8th international conference on Algorithmic number theory*, ANTS-VIII, pages 153–166, Berlin, Heidelberg, 2008. Springer-Verlag.

[BGTZ09] R. Brent, P. Gaudry, E. Thomé, and P. Zimmermann. gf2x. `http://gf2x.gforge.inria.fr/`, 2009.

[BGU07] D. Boucher, W. Geiselmann, and F. Ulmer. Skew-cyclic codes. *Applicable Algebra in Engineering, Communication and Computing*, 18:379–389, 2007.

[BHL10] J. Berthomieu, J. van der Hoeven, and G. Lecerf. Relaxed algorithms for p-adic numbers. *J. Théor. Nombres Bordeaux*, (to appear) 2010. Preliminary version available from `http://hal.archives-ouvertes.fr/hal-00486680/`.

[BKY03] D. Bleichenbacher, A. Kiayias, and M. Yung. Decoding of Interleaved Reed Solomon Codes over Noisy Data. In Jos Baeten, Jan Lenstra, Joachim Parrow, and Gerhard Woeginger, editors, *Automata, Languages and Programming*, volume 2719 of *Lecture Notes in Computer Science*, pages 188–188. Springer Berlin / Heidelberg, 2003.

[Bla83] R.E. Blahut. *Theory and practice of error control codes*. Addison-Wesley Pub. Co., 1983.

[BLQ11] J. Berthomieu, G. Lecerf, and G. Quintin. Polynomial root finding over local rings and application to error correcting codes. `http://hal.inria.fr/hal-00642075`, 2011.

[Bon00] D. Boneh. Finding smooth integers in short intervals using CRT decoding. In *Proceedings of the thirty-second annual ACM symposium on Theory of computing*, STOC '00, pages 265–272, New York, NY, USA, 2000. ACM.

[Bou11] N. Bourbaki. *Algèbre: Chapitre 8*. Springer Verlag, 2011.

[BP94] D. Bini and V. Y. Pan. *Polynomial and matrix computations. Vol. 1. Fundamental algorithms*. Progress in Theoretical Computer Science. Birkhäuser, 1994.

[Bro93] W.C. Brown. *Matrices over commutative rings*. Pure and applied mathematics. M. Dekker, 1993.

[BS05] A. Bostan and É. Schost. Polynomial evaluation and interpolation on special sets of points. *J. Complexity*, 21(4):420–446, 2005.

[BSU08] D. Boucher, P. Solé, and F. Ulmer. Skew constacyclic codes over Galois rings. *Advances in mathematics of communications*, 2(3):273–292, 2008.

[BU09a] D. Boucher and F. Ulmer. Codes as Modules over Skew Polynomial Rings. In Matthew Parker, editor, *Cryptography and Coding*, volume 5921 of *LNCS*, pages 38–55. Springer Berlin / Heidelberg, 2009.

[BU09b] Delphine Boucher and Felix Ulmer. Coding with skew polynomial rings. *Journal of Symbolic Computation*, 44(12):1644–1656, 2009.

[BW86] E. R. Berlekamp and L. R. Welch. Error correction for algebraic block codes, 1986. US Patent 4633470.

[Byr01] E. Byrne. Lifting Decoding Schemes over a Galois Ring. In Serdar Boztas and Igor Shparlinski, editors, *Applied Algebra, Algebraic Algorithms and Error-Correcting Codes*, volume 2227 of *Lecture Notes in Computer Science*, pages 323–332. Springer Berlin / Heidelberg, 2001.

[BZ01] N.S. Babu and K.-H. Zimmermann. Decoding of linear codes over Galois rings. *Information Theory, IEEE Transactions on*, 47(4):1599–1603, may 2001.

[CAY08] N. Chen and Z. A Yan. Complexity Analysis of Reed-Solomon Decoding over GF(2^m) without Using Syndromes. *EURASIP Journal on Wireless Communications and Networking*, 2008, 2008.

[CC86] D. V. Chudnovsky and G. V. Chudnovsky. On expansion of algebraic functions in power and Puiseux series. I. *J. Complexity*, 2(4):271–294, 1986.

[CC87] D. V. Chudnovsky and G. V. Chudnovsky. On expansion of algebraic functions in power and Puiseux series. II. *J. Complexity*, 3(1):1–25, 1987.

[CCN10] P.-L. Cayrel, C. Chabot, and A. Necer. Quasi-cyclic codes as codes over rings of matrices. *Finite Fields and Their Applications*, 16(2):100–115, 2010.

[CGKS95] B. Chor, O. Goldreich, E. Kushilevitz, and M. Sudan. Private information retrieval. In *Foundations of Computer Science, 1995. Proceedings., 36th Annual Symposium on*, pages 41–50, oct 1995.

[CH11] H. Cohn and N. Heninger. Ideal forms of Coppersmith's theorem and Guruswami-Sudan list decoding. In *Proceedings of Innovations in computer science*, 2011.

[Cha11] C. Chabot. Factorisation in $M_\ell(\mathbb{F}_q)[X]$. Construction of quasi-cyclic codes. In *WCC 2011 - Workshop on coding and cryptography*, pages 209–218, Paris, France, apr 2011.

[CK91] D. G. Cantor and E. Kaltofen. On fast multiplication of polynomials over arbitrary algebras. *Acta Inform.*, 28:693–701, 1991.

[CLU09] L. Chaussade, P. Loidreau, and F. Ulmer. Skew codes of prescribed distance or rank. *Designs, Codes and Cryptography*, 50:267–284, 2009.

[Coh46] I. S. Cohen. On the structure and ideal theory of complete local rings. *Trans. Amer. Math. Soc.*, 59:54–106, 1946.

[Coh91] H. Cohen. *Advanced topics in computational algebraic number theory*, volume 193 of *Graduate Texts in Mathematics*. Springer-Verlag, 1991.

[Cop97] D. Coppersmith. Small Solutions to Polynomial Equations, and Low Exponent RSA Vulnerabilities. *J. cryptology*, 10:233–260, 1997.

[CS03] D. Coppersmith and M. Sudan. Reconstructing curves in three (and higher) dimensional space from noisy data. In *Proceedings of the thirty-fifth annual ACM symposium on Theory of computing*, STOC '03, pages 136–142, New York, NY, USA, 2003. ACM.

[CT06] T.M. Cover and J.A. Thomas. *Elements of Information Theory*. Wiley Series in Telecommunications and Signal Processing. Wiley-Interscience, 2006.

[Dum89] I. I. Dumer. Two Decoding Algorithms for Linear Codes. *Problems of Information Transmission*, 25(1):24–32, 1989.

[Duv89] D. Duval. Rational Puiseux expansions. *Compositio Math.*, 70(2):119–154, 1989.

[Eli57] P. Elias. List decoding for noisy channels. Technical report, Research Laboratory of Electronics, Massachusetts Institute of Technology, 1957.

[Eli91] P. Elias. Error-correcting codes for list decoding. *IEEE Trans. Inform. Theory*, 37(1):5–12, jan 1991.

[FOPT10] J.-C. Faugère, A. Otmani, L. Perret, and J.-P. Tillich. Algebraic Cryptanalysis of McEliece Variants with Compact Keys. In Henri Gilbert, editor, *Advances in Cryptology EUROCRYPT 2010*, volume 6110 of *Lecture Notes in Computer Science*, pages 279–298. Springer Berlin / Heidelberg, 2010.

[FS56] A. Fröhlich and J. C. Shepherdson. Effective procedures in field theory. *Philos. Trans. Roy. Soc. London. Ser. A.*, 248:407–432, 1956.

[FS10] C. Fieker and D. Stehlé. Short Bases of Lattices over Number Fields. In G. Hanrot, F. Morain, and E. Thomé, editors, *Algorithmic Number Theory, 9th International Symposium, ANTS-IX, Nancy, France, July 19-23, 2010. Proceedings*, volume 6197 of *Lecture Notes in Computer Science*, pages 157–173. Springer, 2010.

[Für07] M. Fürer. Faster Integer Multiplication. In *Proceedings of the Thirty-Ninth ACM Symposium on Theory of Computing (STOC 2007)*, pages 57–66. ACM, 2007.

[Gao02] S. Gao. A New Algorithm for Decoding Reed-Solomon Codes. In *Communications, Information and Network Security, V. Bhargava, H.V. Poor, V. Tarokh, and S. Yoon*, pages 55–68. Kluwer, 2002.

[Gat84] J. von zur Gathen. Hensel and Newton methods in valuation rings. *Math. Comp.*, 42(166):637–661, 1984.

[GG03] J. von zur Gathen and J. Gerhard. *Modern computer algebra*. Cambridge University Press, second edition, 2003.

[GGR11] P. Gopalan, V. Guruswami, and P. Raghavendra. List Decoding Tensor Products and Interleaved Codes. *SIAM Journal of Computing*, 40(5):1432–1462, 2011.

[GJV03] P. Giorgi, C.-P. Jeannerod, and G. Villard. On the complexity of polynomial matrix computations. In *Proceedings of the 2003 international symposium on Symbolic and algebraic computation*, ISSAC '03, pages 135–142, New York, NY, USA, 2003. ACM.

[GL89] O. Goldreich and L. A. Levin. A hard-core predicate for all one-way functions. In *Proceedings of the twenty-first annual ACM symposium on Theory of computing*, STOC '89, pages 25–32, New York, NY, USA, 1989. ACM.

[Gol07a] I. Goldberg. Improving the Robustness of Private Information Retrieval. In *Security and Privacy, 2007. SP '07. IEEE Symposium on*, pages 131–148, may 2007.

[Gol07b] I. Goldberg. Percy++. Software available from `http://percy.sourceforge.net/`, 2007.

[Gra91] T. Granlund. The GNU Multiple Precision Arithmetic Library, 1991. `http://gmplib.org/`.

[Gra07] Markus Grassl. Bounds on the minimum distance of linear codes and quantum codes. Online available at http://www.codetables.de, 2007. Accessed on 2011-04-19.

[GRS99] O. Goldreich, D. Ron, and M. Sudan. Chinese remaindering with errors. In *Proceedings of the thirty-first annual ACM symposium on Theory of computing*, STOC '99, pages 225–234, New York, NY, USA, 1999. ACM.

[GS98] V. Guruswami and M. Sudan. Improved Decoding of Reed-Solomon and Algebraic-Geometric Codes. *IEEE Trans. Inform. Theory*, 45:1757–1767, 1998.

[GS00] S. Gao and A. Shokrollahi. Computing roots of polynomials over function fields of curves. In D. Joyner, editor, *Proceedings of the Annapolis Conference on Number Theory, Coding Theory, and Cryptography: From Enigma and Geheimschreiber to Quantum Theory*, pages 214–228. Springer-Verlag, 2000.

[GSS00] V. Guruswami, A. Sahai, and M. Sudan. "Soft-decision" decoding of Chinese remainder codes. In *Foundations of Computer Science, 2000. Proceedings. 41st Annual Symposium on*, pages 159–168, 2000.

[GT06] P. Gaudry and E. Thomé. MPFQ : Fast Finite fields, 2006. `http://mpfq.gforge.inria.fr/`.

[Gur03] V. Guruswami. Constructions of codes from number fields. *IEEE Trans. Inform. Theory*, 49(3):594–603, 2003.

[Gur04] V. Guruswami. *List decoding of error-correcting codes: winning thesis of the 2002 ACM doctoral dissertation competition.* Lecture Notes in Computer Science. Springer, 2004.

[Gur10] V. Guruswami. Bridging Shannon and Hamming: List Error-Correction with Optimal Rate. In *Proceedings of ICM 2010 (invited survey)*, aug 2010.

[GV98] M. Greferath and U. Vellbinger. Efficient decoding of $\mathbb{Z}_{p^k}$-linear codes. *IEEE Trans. Inform. Theory*, 44(3):1288–1291, may 1998.

[GW04] M. Grassl and G. White. New good linear codes by special puncturings. In *Information Theory, 2004. ISIT 2004. Proceedings. International Symposium on*, page 454, jun 2004.

[H+02] J. van der Hoeven et al. Mathemagix. Software available from `http://www.mathemagix.org`, 2002.

[Hal01] É. Hallouin. Computing local integral closures. *J. Symbolic Comput.*, 32(3):211–230, 2001.

[Ham50] R. W. Hamming. Error Detecting and Error Correcting Codes. *Bell System Technical Journal*, 29:147–160, apr 1950.

[Har10] W. Hart. Fast Library for Number Theory: An Introduction. In Komei Fukuda, Joris Hoeven, Michael Joswig, and Nobuki Takayama, editors, *Mathematical Software ICMS 2010*, volume 6327 of *Lecture Notes in Computer Science*, pages 88–91. Springer Berlin / Heidelberg, 2010. `http://www.flintlib.org/`.

[HM87] J.-P. Henry and M. Merle. Complexity of Computation of Embedded Resolution of Algebraic Curves. In *Proceedings of Eurocal 87*, volume 378 of *LNCS*, pages 381–390. Springer-Verlag, 1987.

[IPE97] J.C. Interlando, Jr. Palazzo, R., and M. Elia. On the decoding of Reed-Solomon and BCH codes over integer residue rings. *IEEE Trans. Inform. Theory*, 43(3):1013–1021, may 1997.

[Iwa05] M. Iwami. Extension of expansion base algorithm for multivariate analytic factorization including the case of singular leading coefficient. *SIGSAM Bull.*, 39(4):122–126, 2005.

[Joy00] D. Joyner. *Coding theory and cryptography: from Enigma and Geheimschreiber to quantum theory*. Springer-Verlag, 2000.

[Jus76] J. Justesen. On the complexity of decoding Reed-Solomon codes (Corresp.). *IEEE Trans. Inform. Theory*, 22(2):237–238, March 1976.

[JV05] C.P. Jeannerod and G. Villard. Essentially optimal computation of the inverse of generic polynomial matrices. *Journal of Complexity*, 21(1):72–86, 2005. Foundations of Computational Mathematics Conference 2002.

[JV06] C.-P. Jeannerod and G. Villard. Asymptotically fast polynomial matrix algorithms for multivariable systems. *International Journal of Control*, 79(11):1359–1367, 2006.

[Ked01] K. S. Kedlaya. The algebraic closure of the power series field in positive characteristic. *Proc. Amer. Math. Soc.*, 129(12):3461–3470, 2001.

[Köt96] R. Kötter. *On Algebraic Decoding of Algebraic-Geometric and Cycling Codes*. PhD thesis, Linköping University, Sweden, 1996.

[Kro82] L. Kronecker. Grundzüge einer arithmetischen Theorie der algebraischen Grössen. (Abdruck einer Festschrift zu Herrn E. E. Kummers Doctor-Jubiläum, 10. September 1881.). *Journal für die reine und angewandte Mathematik*, 92:1–122, 1882.

[Kuo89] T. C. Kuo. Generalized Newton-Puiseux theory and Hensel's lemma in $\mathbf{C}[\![x, y]\!]$. *Canad. J. Math.*, 41(6):1101–1116, 1989.

[KV03] R. Kötter and A. Vardy. Algebraic soft-decision decoding of Reed-Solomon codes. *IEEE Trans. Inform. Theory*, 49(11):2809–2825, 2003.

[Lan02] S. Lang. *Algebra*, volume 211 of *Graduate Texts in Mathematics*. Springer-Verlag, third edition, 2002.

[Laz65] M Lazard. Graduations, filtrations, valuations. *Publications Mathématiques de L'IHÉS*, 26:15–43, 1965.

[LDW94] Y. X. Li, R. H. Deng, and X. M. Wang. On the equivalence of McEliece's and Niederreiter's public-key cryptosystems. *IEEE Trans. Inform. Theory*, 40(1):271–273, January 1994.

[Lec08] G. Lecerf. Fast Separable Factorization and Applications. *Appl. Algebra Engrg. Comm. Comput.*, 19(2):135–160, 2008.

[Len86] H. Lenstra. Codes from algebraic number fields. In M. Hazewinkel, J.K. Lenstra, and L.G. L.T. Meertens, editors, *Mathematics and computer science II, Fundamental contributions in the Netherlands since 1945*, volume 4 of *CWI Monograph*, pages 94–104, North-Holland, Amsterdam, 1986.

[LF01] K. Lally and P. Fitzpatrick. Algebraic structure of quasicyclic codes. *Discrete Applied Mathematics*, 111(1–2):157–175, 2001.

[LLL82] A. K. Lenstra, H. W. Lenstra, and L. Lászlo. Factoring polynomials with rational coefficients. *Math. Ann.*, 261:515–534, 1982.

[LS01] S. Ling and P. Solé. On the algebraic structure of quasi-cyclic codes .I. Finite fields. *IEEE Trans. Inform. Theory*, 47(7):2751–2760, nov 2001.

[LS03] S. Ling and P. Solé. Good self-dual quasi-cyclic codes exist. *IEEE Trans. Inform. Theory*, 49(4):1052–1053, april 2003.

[Man76] D. Mandelbaum. On a class of arithmetic codes and a decoding algorithm (Corresp.). *IEEE Trans. Inform. Theory*, 22(1):85–88, jan 1976.

[Map12] Maplesoft. Maple. `http://www.maplesoft.com/`, 2012.

[Mas69] J. Massey. Shift-register synthesis and BCH decoding. *IEEE Trans. Inform. Theory*, 15(1):122–127, jan 1969.

[Mat80] H. Matsumura. *Commutative Algebra*. Mathematics Lecture Note Series. Benjamin/Cummings Publishing Company, 1980.

[McE78] R. McEliece. A Public-Key Cryptosystem Based On Algebraic Coding Theory. *Deep Space Network Progress Report*, 44:114–116, 1978.

[McE03] R. J. McEliece. The Guruswami-Sudan Decoding Algorithm for Reed-Solomon Codes, 2003. `http://www.ee.caltech.edu/EE/Faculty/rjm/papers/RSD-JPL.pdf`.

[Moo05] T. K. Moon. *Error Correction Coding: Mathematical Methods and Algorithms*. Wiley-Interscience, 2005.

[MRS01] J.C. McConnell, J.C. Robson, and L.W. Small. *Noncommutative Noetherian rings*. Graduate studies in mathematics. American Mathematical Society, 2001.

[MS81] R. J. McEliece and D. V. Sarwate. On sharing secrets and Reed-Solomon codes. *Commun. ACM*, 24(9):583–584, September 1981.

[MS86a] F.J. MacWilliams and N.J.A. Sloane. *The theory of error-correcting codes*. North-Holland mathematical library. North-Holland, 1986.

[MS86b] R. McEliece and L. Swanson. On the decoder error probability for Reed-Solomon codes (Corresp.). *IEEE Trans. Inform. Theory*, 32(5):701–703, sep 1986.

[MS03] T. Mulders and A. Storjohann. On lattice reduction for polynomial matrices. *J. Symbolic Comput.*, 35(4):377–401, 2003.

[Neu99] J. Neukirch. *Algebraic number theory*. Comprehensive Studies in Mathematics. Springer-Verlag, 1999. ISBN 3-540-65399-6.

[NH00] Rasmus R. Nielsen and T. Hoeholdt. Decoding Reed-Solomon codes beyond half the minimum distance. In Johannes Buchmann, Tom Hoeholdt, Henning Stichtenoth, and Horacio Tapia Recillas, editors, *Coding Theory, Cryptography and Related Areas*. Springer-Verlag, April 2000.

[Nie86] H. Niederreiter. Knapsack-type cryptosystems and algebraic coding theory. *Problems of Control and Information Theory*, 15(2):159–166, 1986.

[Nor99] G. Norton. On Minimal Realization Over a Finite Chain Ring. *Designs, Codes and Cryptography*, 16:161–178, 1999.

[NSM00] G.H. Norton and A. Salagean Mandache. On the Key Equation Over a Commutative Ring. *Designs, Codes and Cryptography*, 20:125–141, 2000.

[OS99] V. Olshevsky and M. A. Shokrollahi. A displacement approach to efficient decoding of algebraic-geometric codes. In *Proceedings of the thirty-first annual ACM symposium on Theory of computing*, STOC '99, pages 235–244, New York, NY, USA, 1999. ACM.

[OSB12] F. Oggier, P. Sole, and J.-C. Belfiore. Codes Over Matrix Rings for Space-Time Coded Modulations. *IEEE Trans. Inform. Theory*, 58(2):734–746, February 2012.

[PAR11] The PARI Group, Bordeaux. *PARI/GP, version `2.5.0`*, 2011. available from `http://pari.math.u-bordeaux.fr/`.

[Pet60] W. Peterson. Encoding and error-correction procedures for the Bose-Chaudhuri codes. *Information Theory, IRE Transactions on*, 6(4):459–470, sep 1960.

[PR10] A. Poteaux and M. Rybowicz. Complexity Bounds for the rational Newton-Puiseux Algorithm over Finite Fields. *Appl. Algebra Engrg. Comm. Comput.*, (to appear) 2010.

[PW72] W.W. Peterson and E.J. Weldon. *Error-correcting codes*. MIT Press, 1972.

[Rag69] R. Raghavendran. Finite associative rings. *Compositio Math.*, 21:195–229, 1969.

[Rot06] R. Roth. *Introduction to coding theory*. Cambridge University Press, 2006.

[RR98] R. M. Roth and G. Ruckenstein. Efficient decoding of Reed-Solomon codes beyond half the minimum distance. In *IEEE Trans. Inform. Theory*, page 56, 1998.

[RS60] I. S. Reed and G. Solomon. Polynomial Codes Over Certain Finite Fields. *Journal of the Society for Industrial and Applied Mathematics*, 8(2):300–304, 1960.

[RU10] A. Rudra and S. Uurtamo. Two Theorems on List Decoding. In Maria Serna, Ronen Shaltiel, Klaus Jansen, and José Rolim, editors, *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques*, volume 6302 of *Lecture Notes in Computer Science*, pages 696–709. Springer Berlin / Heidelberg, 2010.

[Ser62] J.-P. Serre. *Corps locaux*. Number ns 1296 à 1297 in Actualités scientifiques et industrielles. Hermann, 1962.

[Sha48] C. E. Shannon. A mathematical theory of communication. *Bell System Technical Journal*, 5:3–55, 1948.

[Sha79] A. Shamir. How to share a secret. *Commun. ACM*, 22(11):612–613, November 1979.

[Sho90] V. Shoup. NTL: A Library for doing Number Theory, 1990. `http://www.shoup.net/ntl/index.html`.

[Sid94] V. M. Sidelnikov. Decoding Reed-Solomon Codes Beyond $(d-1)/2$ and Zeros of Multivariate Polynomials. *Problems of Information Transmission*, 30(1):44–59, 1994.

[SKHN75] Y. Sugiyama, M. Kasahara, S. Hirasawa, and T. Namekawa. A method for solving key equation for decoding goppa codes. *Information and Control*, 27(1):87–99, 1975.

[Smi81] P.F. Smith. Injective modules and prime ideals. *Communications in Algebra*, 9(9):989–999, 1981.

[SS71] A. Schönhage and V. Strassen. Schnelle Multiplikation grosser Zahlen. *Computing*, 7:281–292, 1971.

[Sud97a] M. Sudan. Decoding of Reed-Solomon Codes beyond the Error-Correction Bound. *Journal of Complexity*, 13(1):180–193, 1997.

[Sud97b] M. Sudan. Decoding Reed-Solomon codes beyond the error-correction diameter. In *the 35th Annual Allerton Conference on Communication, Control and Computing*, pages 215–224, 1997.

[SV05] A. Storjohann and G. Villard. Computing the rank and a small nullspace basis of a polynomial matrix. In *Proceedings of the 2005 international symposium on Symbolic and algebraic computation*, ISSAC '05, pages 309–316, New York, NY, USA, 2005. ACM.

[TERH88] T. K. Truong, W. L. Eastman, I. S. Reed, and I. S. Hsu. Simplified procedure for correcting both errors and erasures of Reed-Solomon code using Euclidean algorithm. *IEEE Proc. Comput. and Digit. Tech.*, 135(6):318–324, 1988.

[UL10] V. G. Umaña and G. Leander. Practical Key Recovery Attacks On Two McEliece Variants. In Carlos Cid and Jean-Charles Faugére, editors, *Proceedings of the Second International Conference on Symbolic Computation and Cryptography*, pages 27–44, June 2010.

[VVO89] S.A. Vanstone and P.C. Van Oorschot. *An Introduction to Error Correcting Codes With Applications.* Kluwer International Series in Engineering and Computer Science. Kluwer Academic Publishers, 1989.

[VW99] J.F. Veloch and J.L. Walker. Codes over rings from curves of higher genus. *IEEE Trans. Inform. Theory*, 45(6):1768–1776, sep 1999.

[Wal78] R. J. Walker. *Algebraic curves.* Springer-Verlag, New York, 1978.

[Wal99a] J. L. Walker. Algebraic geometric codes over rings. *J. Pure Appl. Algebra*, 144(1):91–110, 1999.

[Wal99b] P. G. Walsh. On the complexity of rational Puiseux expansions. *Pacific J. Math.*, 188(2):369–387, 1999.

[Wal00] P. G. Walsh. A polynomial-time complexity bound for the computation of the singular part of a Puiseux expansion of an algebraic function. *Math. Comp.*, 69(231):1167–1182, 2000.

[WB99] S.B. Wicker and V.K. Bhargava. *Reed-Solomon Codes and Their Applications.* John Wiley & Sons, 1999.

[WB08] J. Walker and K. Bartley. *Algebraic Geometric Codes over Rings.* Series on Coding Theory and Cryptology. World Scientific, 2008.

[Woo99] J.A Wood. Duality for modules over finite rings and applications to coding theory. *American Journal of Mathematics*, 121(3):555–575, 1999.

[Woz58] J. M. Wozencraft. List Decoding. Quarterly Progress Report, Research Laboratory of Electronics, MIT, 1958.

[ZP81] V. V. Zyablov and M. S. Pinsker. List Cascade Decoding. *Problems of Information Transmission*, 17(4):29–34, 1981. (in Russian) pp. 236–240 (in English), 1982.

List of Symbols

$\mathbb{N}$	natural numbers
$\mathbb{Z}$	integers
$\mathbb{Q}$	rational numbers
$\mathbb{R}$	real numbers
$\mathbb{C}$	complex numbers
$\mathbb{F}_q$	finite fields with q elements
$\mathrm{GR}(p^r, s)$	Galois ring with p^{rs} elements of characteristic p^r
$\mathbb{Z}_p$	p-adic integers
$\mathbb{Z}_{p^s}$	unramified extension of the p-adic integers
κ	field
$\kappa[[t]]$	power series in t over κ
$\kappa[[t_1, \ldots, t_r]]$	power series in $t_1, ..., t_r$ over κ
$\mathcal{C}$	error correcting code
$\mathfrak{p}$	prime ideal
$\mathfrak{m}$	maximal ideal
$\mathcal{O}_K$	algebraic integers in the number field K
$Z(A)$	center of the ring A
$A^\times$	group of units of the ring A
$\mathbb{A}^2(\mathbb{F}_q)$	affine plane over $\mathbb{F}_q$
$A[X]_{<k}$	polynomials of degree at most $k-1$ over the ring A
$\lfloor x \rfloor$	largest previous integer of x
$\lceil x \rceil$	smallest integer following x
$\binom{n}{k}$	binomial coefficient for $n, k \in \mathbb{N}$
$\mathrm{RS}_A(x, k)$	Reed-Solomon code of support x and dimension k over the ring A
$\mathrm{RS}_A(n, k)$	Reed-Solomon code of length n and dimension k over the ring A
$\mathrm{GRS}_A(n, k)$	generalized Reed-Solomon code of length n and dimension k over the ring A
$\mathrm{GRS}_A(v, x, k)$	generalized Reed-Solomon code of weight v, support x and dimension k over the ring A
$\text{Q-BCH}(m, \ell, \delta, A)$	ℓ-quasi-BCH code of length $m\ell$ and designed minimum distance δ with respect to A
$[n, k, d]_A$	free linear code over A of length n, dimension k and dimension d

$M_\ell(\mathbb{F}_q)$	square $\ell \times \ell$ matrices over $\mathbb{F}_q$
$\mathrm{GL}_\ell(\mathbb{F}_{q^{s\ell}})$	group of invertible $\ell \times \ell$ matrices over $\mathbb{F}_{q^{s\ell}}$
$\mathsf{M}(n)$	cost for multiplying two polynomials of degree n
$\mathsf{I}(n)$	cost for multiplying two integers of bit-size n
ω	feasible linear algebra exponent
$\langle f \rangle$	module spanned by the components of f

Index

Abstract

This thesis studies the algorithmic techniques of list decoding, first proposed by Guruswami and Sudan in 1998, in the context of Reed-Solomon codes over finite rings. Two approaches are considered. First we adapt the Guruswami-Sudan (GS) list decoding algorithm to generalized Reed-Solomon (GRS) codes over finite rings with identity. We study in details the complexities of the algorithms for GRS codes over Galois rings and truncated power series rings. Then we explore more deeply a lifting technique for list decoding. We show that the latter technique is able to correct more error patterns than the original GS list decoding algorithm. We apply the technique to GRS code over Galois rings and truncated power series rings and show that the algorithms coming from this technique have a lower complexity than the original GS algorithm. We show that it can be easily adapted for interleaved Reed-Solomon codes. Finally we present the complete implementation in C and C++ of the list decoding algorithms studied in this thesis. All the needed subroutines, such as univariate polynomial root finding algorithms, finite fields and rings arithmetic, are also presented. Independently, this manuscript contains other work produced during the thesis. We study quasi cyclic codes in details and show that they are in one-to-one correspondence with left principal ideal of a certain matrix ring. Then we adapt the GS framework for ideal based codes to number fields codes and provide a list decoding algorithm for the latter.

Résumé

Cette thèse porte sur l'algorithmique des techniques de décodage en liste, initiée par Guruswami et Sudan en 1998, dans le contexte des codes de Reed-Solomon sur les anneaux finis. Deux approches sont considérées. Dans un premier temps, nous adaptons l'algorithme de décodage en liste de Guruswami-Sudan aux codes de Reed-Solomon généralisés sur les anneaux finis. Nous étudions en détails les complexités de l'algorithme pour les anneaux de Galois et les anneaux de séries tronquées. Dans un deuxième temps nous approfondissons l'étude d'une technique de remontée pour le décodage en liste. Nous montrons que cette dernière permet de corriger davantage de motifs d'erreurs que la technique de Guruswami-Sudan originale. Nous appliquons ensuite cette même technique aux codes de Reed-Solomon généralisés sur les anneaux de Galois et les anneaux de séries tronquées et obtenons de meilleures bornes de complexités. Enfin nous présentons l'implantation des algorithmes en C et C++ des algorithmes de décodage en liste étudiés au cours de cette thèse. Tous les sous-algorithmes nécessaires au décodage en liste, comme la recherche de racines pour les polynômes univariés, l'arithmétique des corps et anneaux finis sont aussi présentés. Indépendamment, ce manuscrit contient d'autres travaux sur les codes quasi-cycliques. Nous prouvons qu'ils sont en correspondance biunivoque avec les idéaux à gauche d'un certain anneaux de matrices. Enfin nous adaptons le cadre proposé par Guruswami et Sudan pour les codes à base d'ideaux aux codes construits à l'aide des corps de nombres. Nous fournissons un algorithme de décodage en liste dans ce contexte.